Image Filtering

EECS 442 – Jeong Joon Park
Winter 2024, University of Michigan

Administrivia

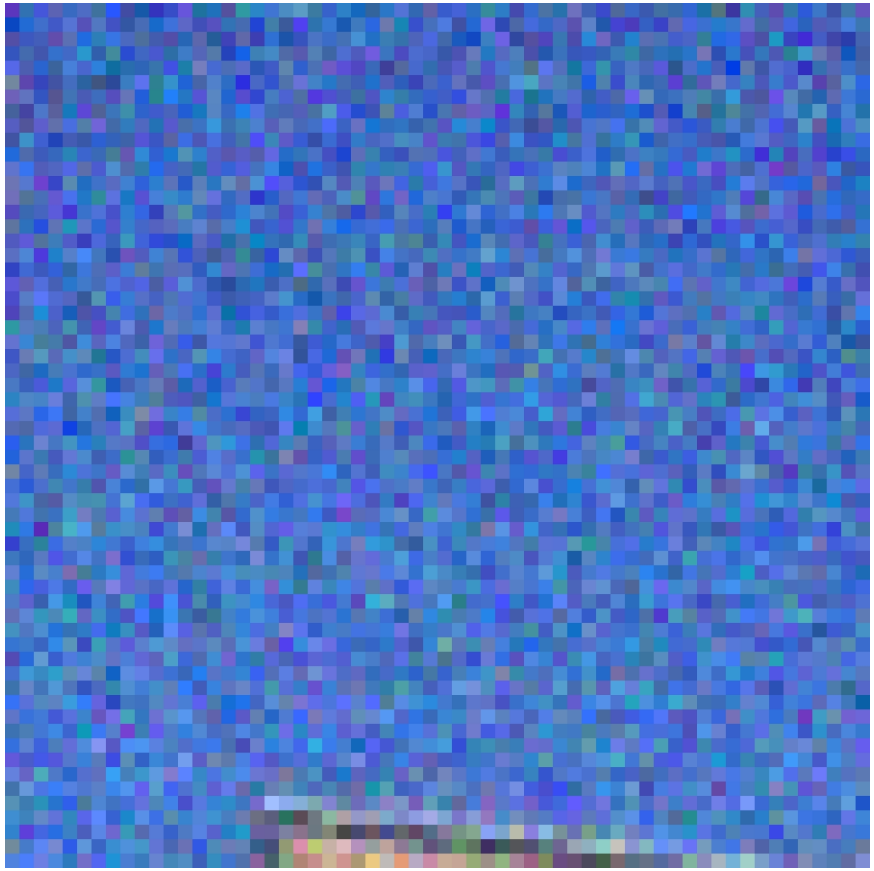
- HW1 due Wednesday at 11:59pm (although you do have late tokens…)
  - TAs work 9 AM-5 PM. So don't expect answers late
- Next few classes and homeworks: building momentum towards stuff
- Good news: probably won't be as tricky implementation wise now that you've seen the bugs
- Bad news: will be trickier conceptual-wise

Administrivia

- Piazza and Office Hours

1. Don't post code publicly (Or pictures)
2. Ask *specific questions* about small code snippets (1 or 2 lines)
   - We cannot debug your code. 7TAs for 280 students
   - Prepare specific questions for OH
3. No "does this look right?" questions
4. Show respect. No “Hey, can you help on this, lol?”
5. Carefully Read the Announcement

Let’s Take An Image



Let's Fix Things

- We have noise in our image
- Let's replace each pixel with a *weighted* average of its neighborhood
- Weights are *filter kernel*

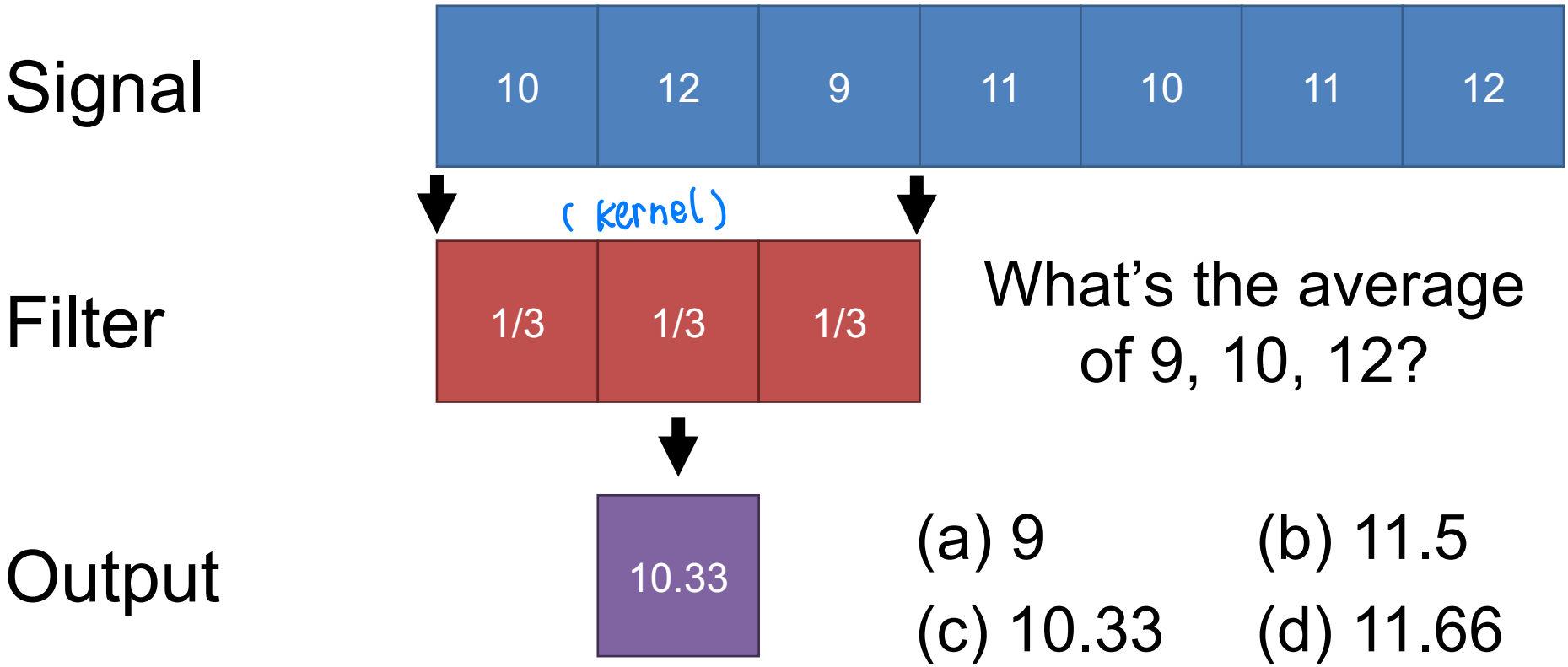
	Out	

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Slide Credit: D. Lowe

np.dot(input, filter)

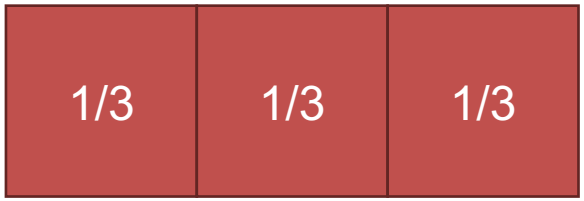
1D Case


Signal
10
12
9
11
10
11
12
(kernel)
Filter
1/3
1/3
1/3
What's the average of 9, 10, 12?
Output
10.33
(a) 9
(b) 11.5
(c) 10.33
(d) 11.66

1D Case

Signal

Filter



Output

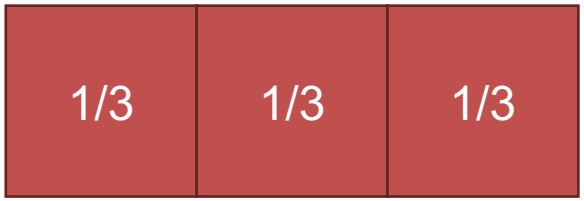
Done! Next?

1D Case

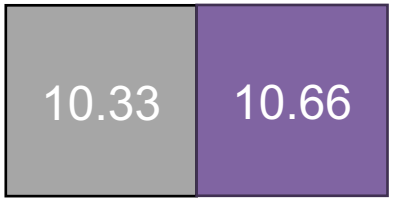
(1) 10.66 (2) 9.33
(3) 14.2 (4) 11.33

Signal

Filter



Output

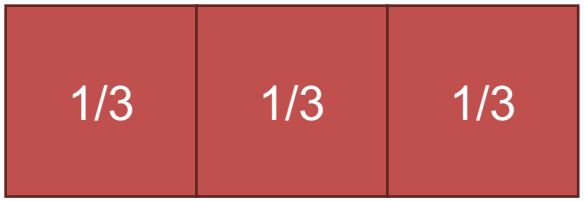


1D Case

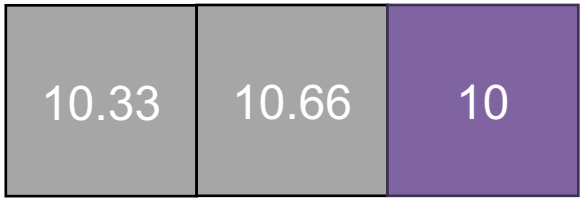
(1) 10.33 (2) 11.33
(3) 10 (4) 9.1

Signal

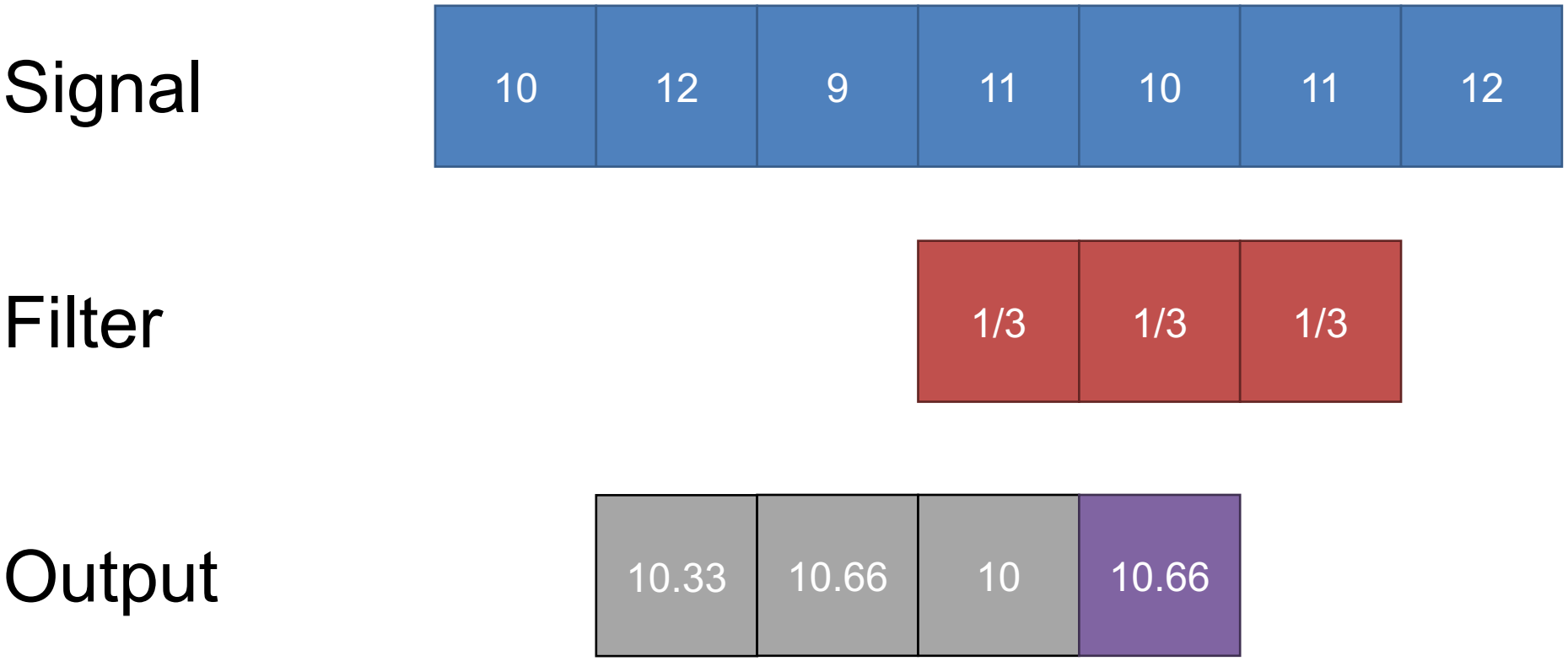
Filter



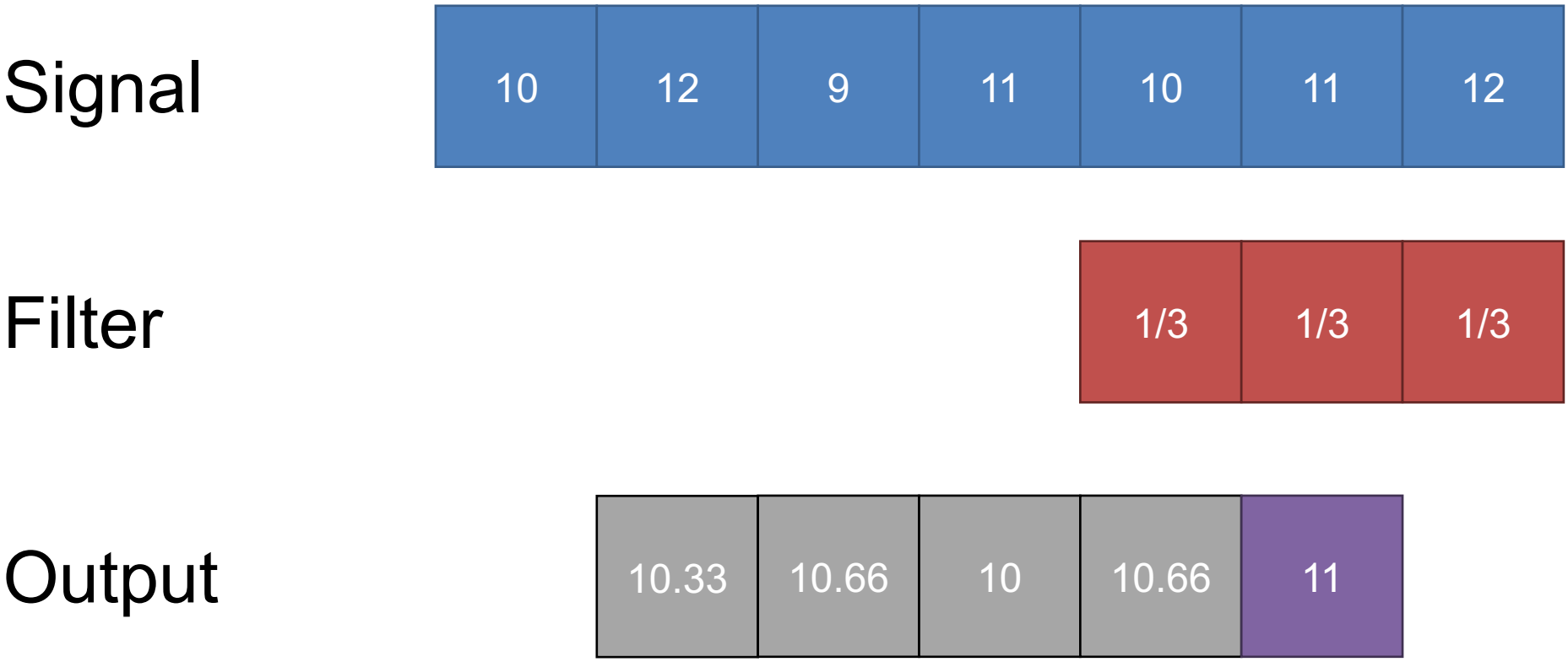
Output



1D Case

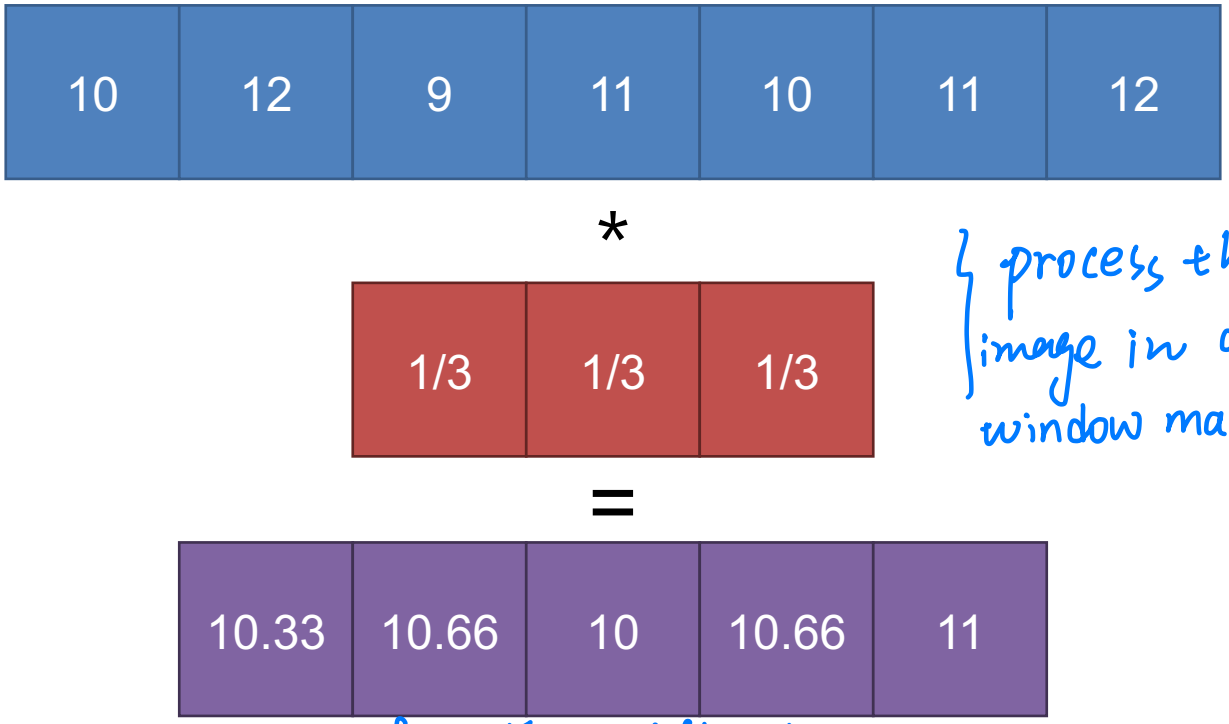

Signal
10
12
9
11
10
11
12
Filter
1/3
1/3
1/3
Output
10.33
10.66
10
10.66

1D Case


Signal
10
12
9
11
10
11
12
Filter
1/3
1/3
1/3
Output
10.33
10.66
10
10.66
11

This concept is called sliding window.

1D Case



process the image in a sliding window manner

before : 7 After : 5

You lose pixels (maybe…)

Filter “sees” only a few pixels at a time

Slider window Here, only see 3 at each time.

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

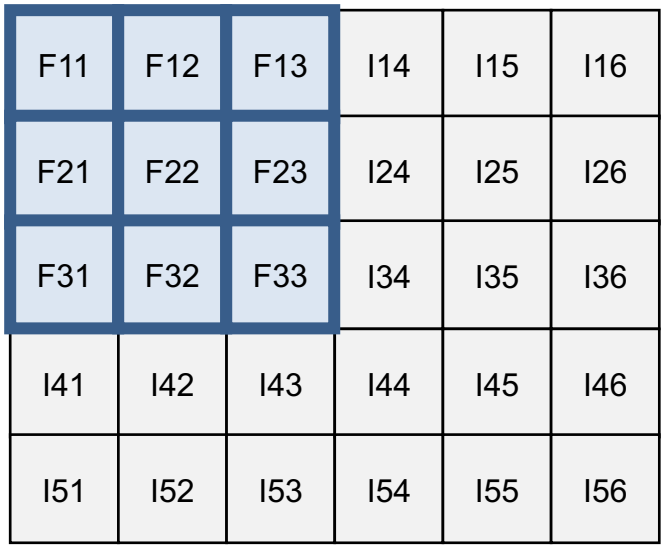
F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

Applying a Linear Filter

Input & Filter



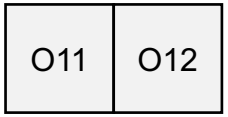
Output

$$O11 = I11*F11 + I12*F12 + \ldots + I33*F33$$

Applying a Linear Filter

Input & Filter

Output



$$O12 = I12*F11 + I13*F12 + \ldots + I34*F33$$

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

How many times can we apply a 3x3 filter to a 5x6 image?

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

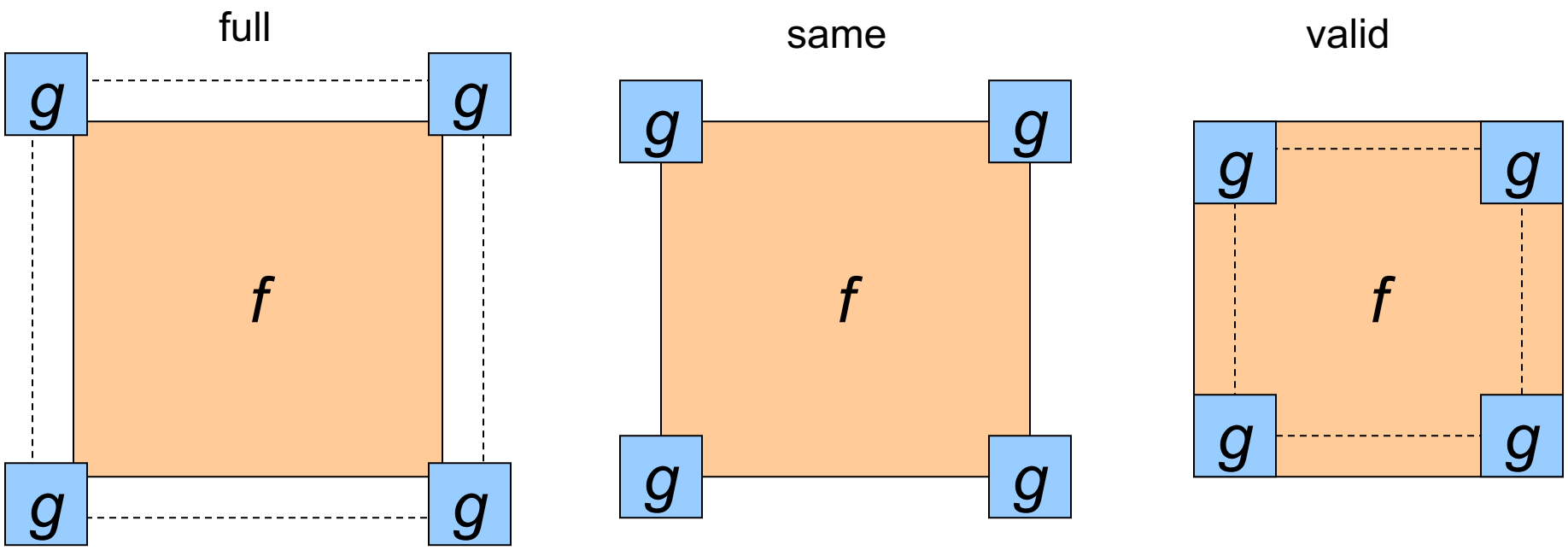
O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

$$Oij = Iij*F11 + Ii(j+1)*F12 + \ldots + I(i+2)(j+2)*F33$$

Painful Details – Edge Cases

Filtering doesn't keep the whole image.
Suppose **f** is the image and **g** the filter.

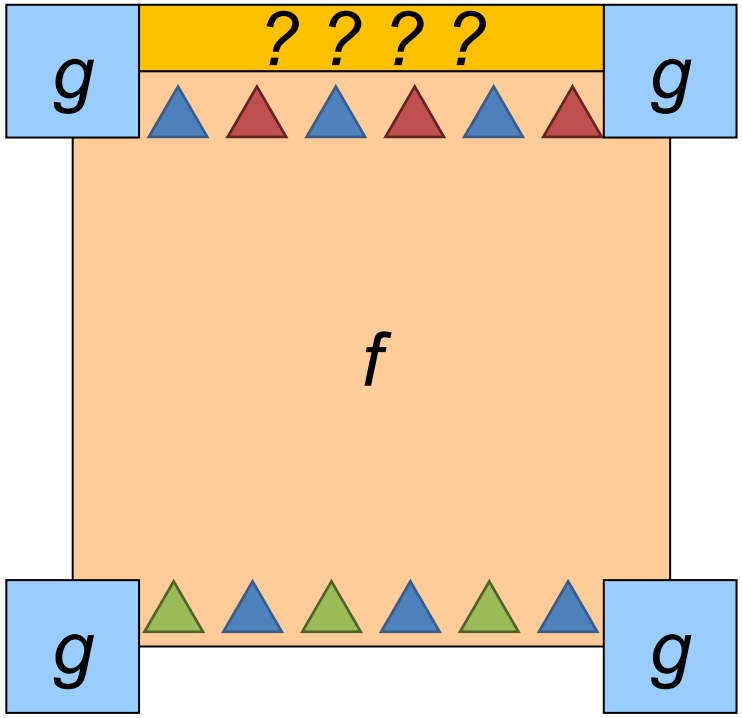
Full – any part of g touches f. **Same** – same size as f;
Valid – only when filter doesn't fall off edge.



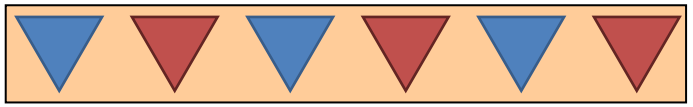
f/g Diagram Credit: D. Lowe

Painful Details – Edge Cases

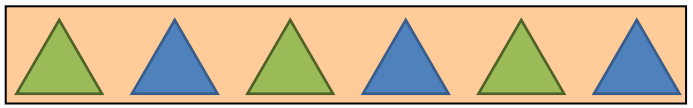
What to about the “?” region?



Symm: fold sides over



Circular/Wrap: wrap around



pad/fill: add value, often 0

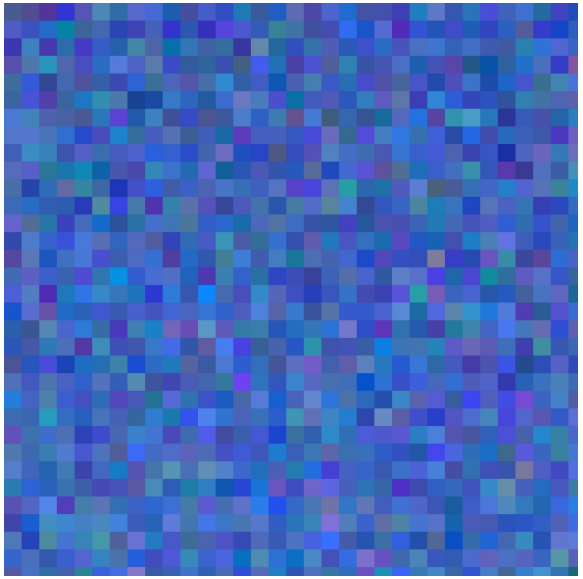
f/g Diagram Credit: D. Lowe

Painful Details – Does it Matter?

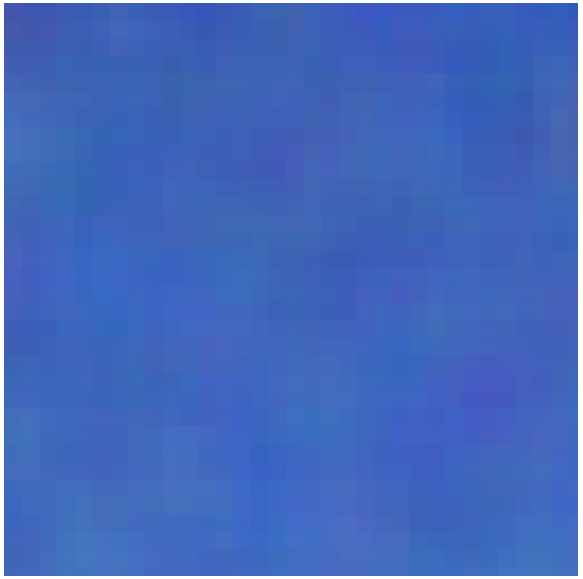
(I've applied the filter per-color channel)

Which padding did I use and *why*?

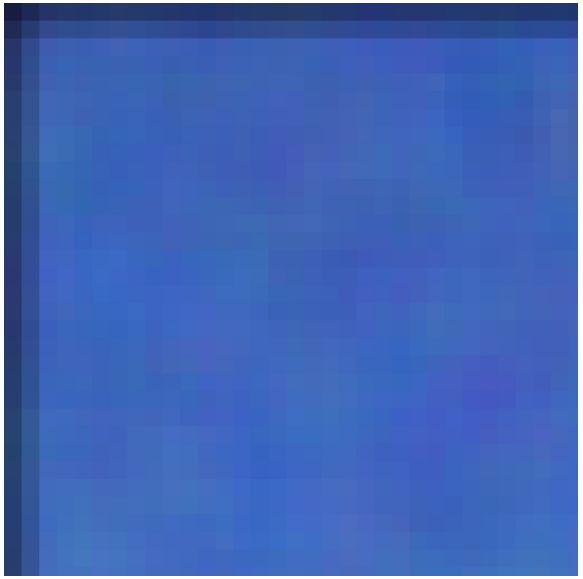
Input Image



Filtered ???



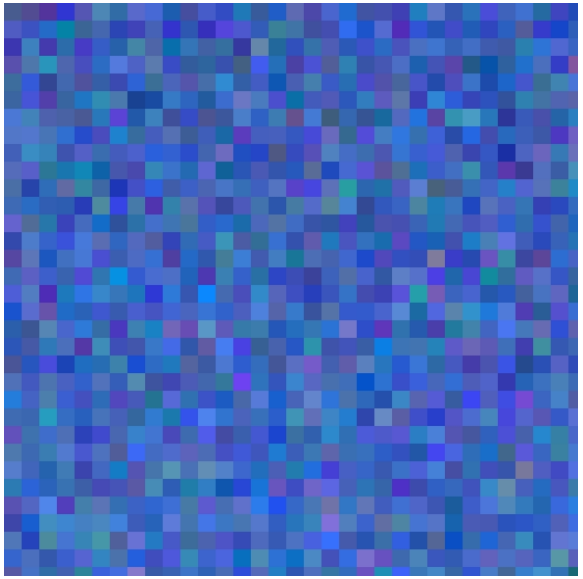
Filtered ???



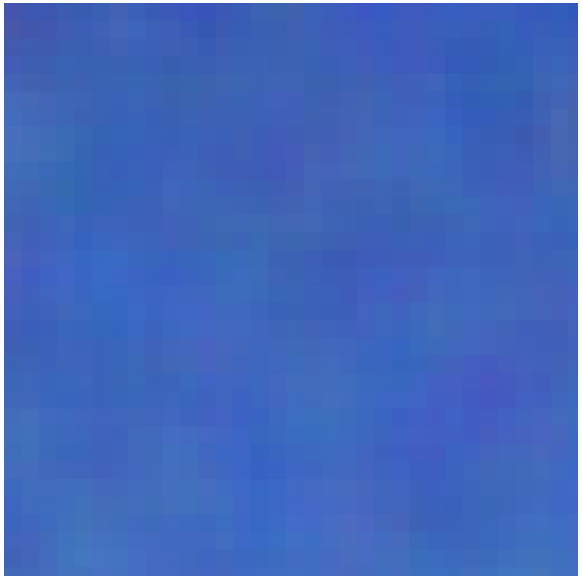
Painful Details – Does it Matter?

(I’ve applied the filter per-color channel)

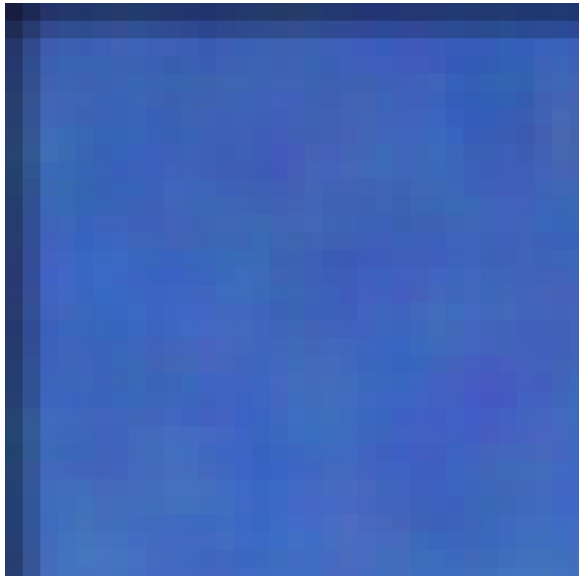
Input Image



Filtered Symm Pad



Filtered Zero Pad



Practice with Linear Filters

Original

0	0	0
0	1	0
0	0	0

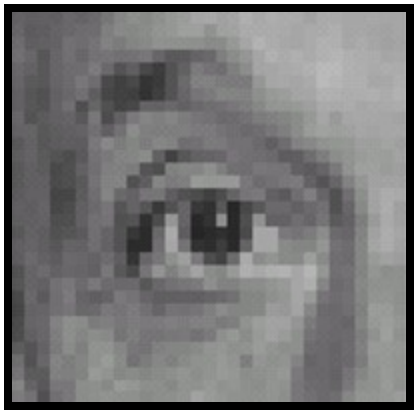
?

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	0	0
0	1	0
0	0	0



The Same!

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	0	0
0	0	1
0	0	0

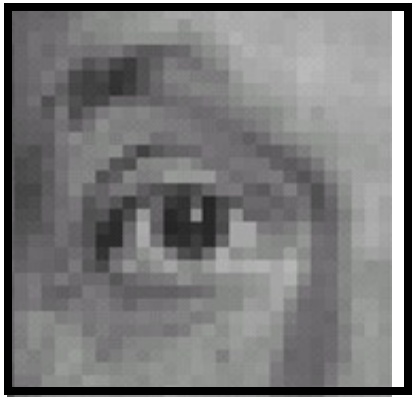
?

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	0	0
0	0	1
0	0	0



Shifted
LEFT
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	1	0
0	0	0
0	0	0

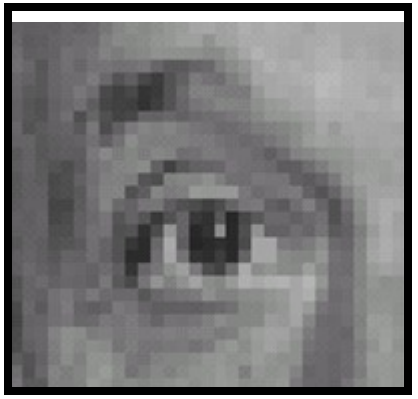
?

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	1	0
0	0	0
0	0	0



Shifted
DOWN
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters

Original

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

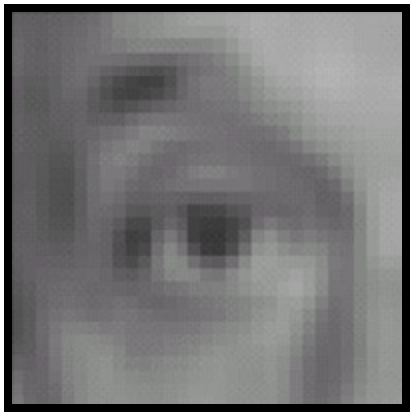
?

Slide Credit: D. Lowe

Practice with Linear Filters

Original

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



Blur
(Box Filter)

Slide Credit: D. Lowe

Practice with Linear Filters

Original

0	0	0
0	2	0
0	0	0

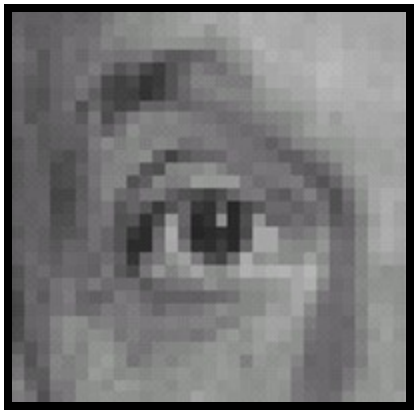
-

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

?

Slide Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	2	0
0	0	0

-

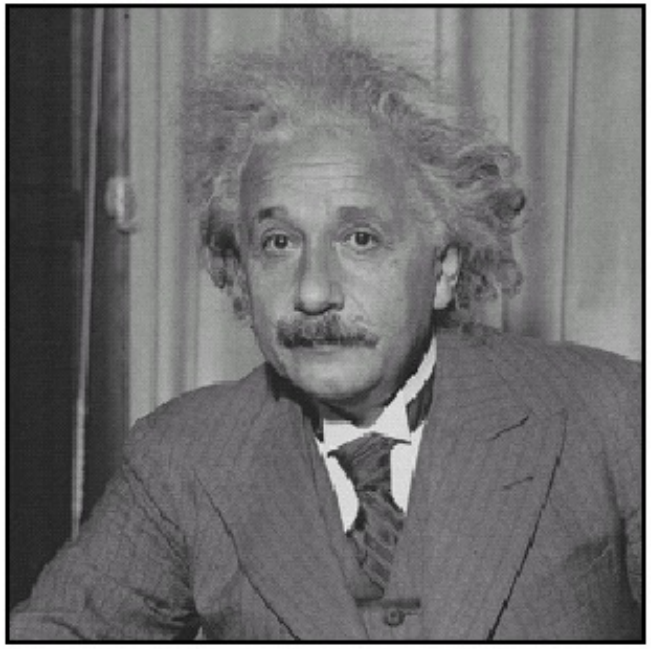
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Sharpened
(Acccentuates difference from local average)

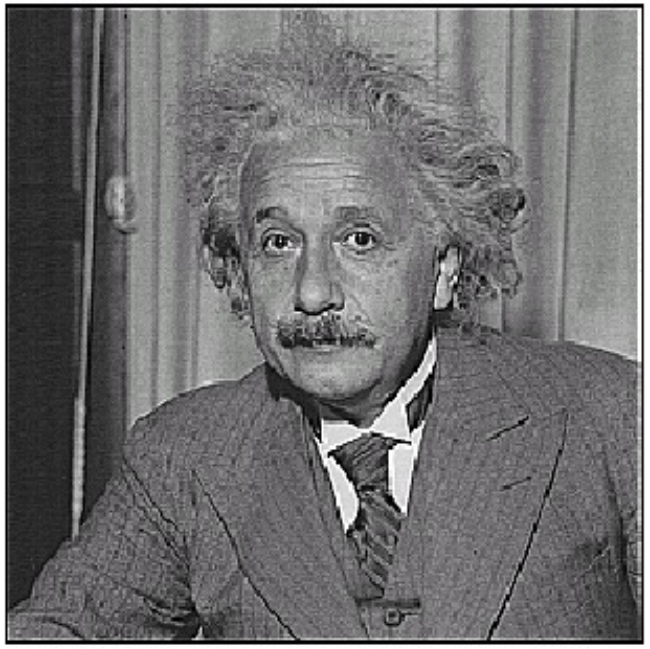
emphasizing the high-frequency details of this image ↓ gives me a more sharpened image

Slide Credit: D. Lowe

Sharpening



before



after

Slide Credit: D. Lowe

Properties – Linear

Input

I11	I12	I13	I14
I21	I22	I23	I24
I31	I32	I33	I34
I41	I42	I43	I44

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12
O21	O22

We want to do by simple matrix multiplication.

Flatten Input and multiply some matrix W

Can exactly mimic the sliding window filtering

Linear filtering is linear!

Properties – Linear

Input

I11	I12	I13	I14
I21	I22	I23	I24
I31	I32	I33	I34
I41	I42	I43	I44

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12
O21	O22

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & w_{1,6} & w_{1,7} & w_{1,8} & w_{1,9} & w_{1,10} & w_{1,11} & w_{1,12} & w_{1,13} & w_{1,14} & w_{1,15} & w_{1,16} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & w_{2,6} & w_{2,7} & w_{2,8} & w_{2,9} & w_{2,10} & w_{2,11} & w_{2,12} & w_{2,13} & w_{2,14} & w_{2,15} & w_{2,16} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & w_{3,6} & w_{3,7} & w_{3,8} & w_{3,9} & w_{3,10} & w_{3,11} & w_{3,12} & w_{3,13} & w_{3,14} & w_{3,15} & w_{3,16} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} & w_{4,7} & w_{4,8} & w_{4,9} & w_{4,10} & w_{4,11} & w_{4,12} & w_{4,13} & w_{4,14} & w_{4,15} & w_{4,16} \end{bmatrix}$$

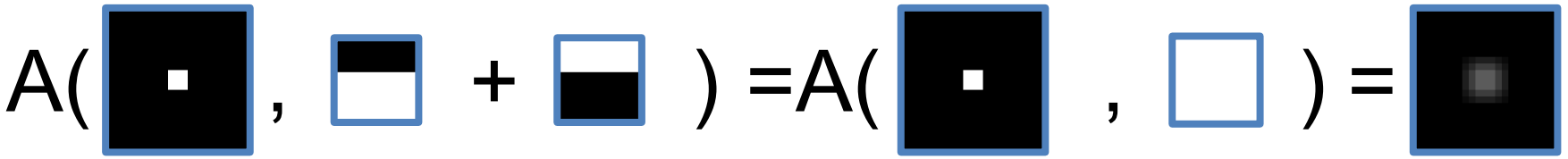
W

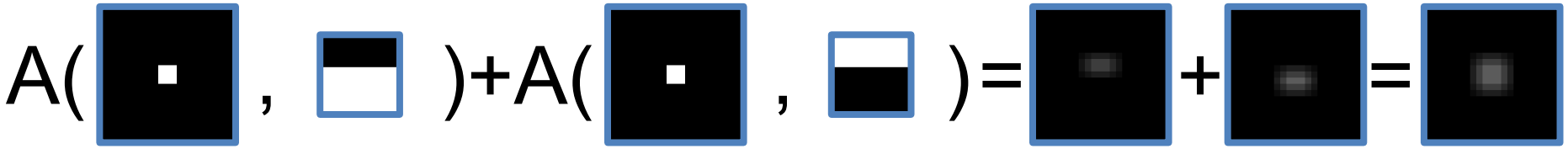
Properties – Linear

Assume: I image f1, f2 filters

Linear: apply(I,f1+f2) = apply(I,f1) + apply(I,f2)

I is a white box on black, and f1, f2 are rectangles





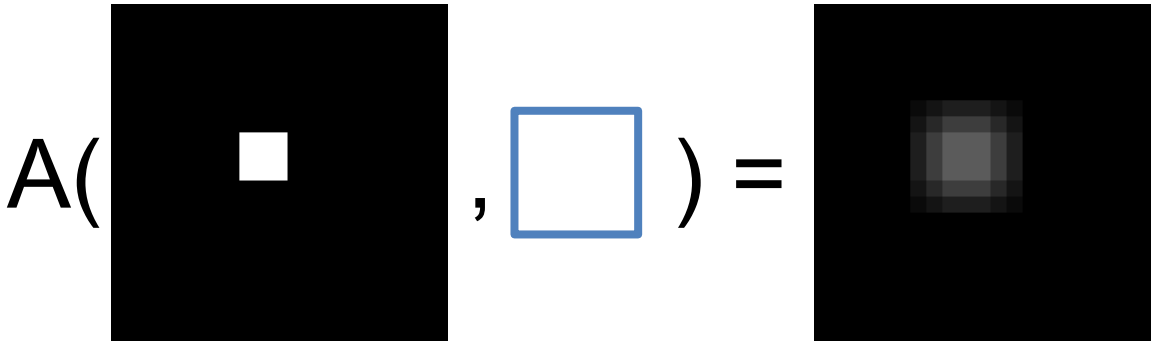
Note: I am showing filters un-normalized and blown up. They're a smaller box filter (i.e., each entry is 1/(size^2))

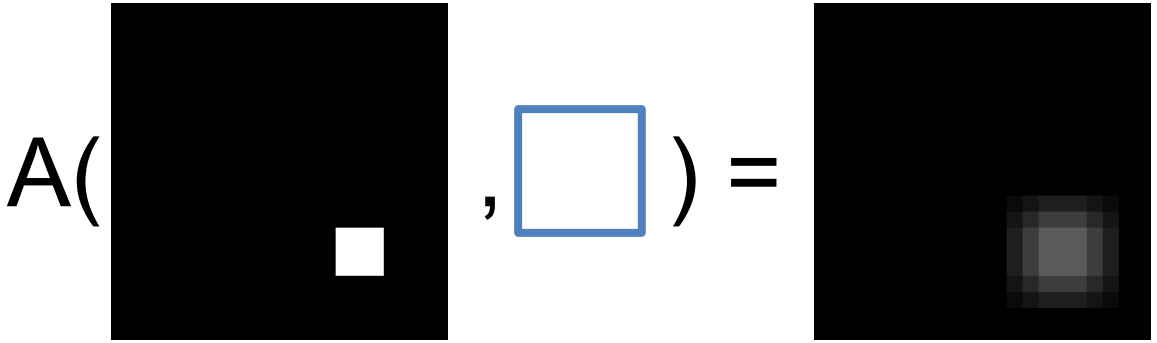
Properties – Shift-Invariant

Assume: I image, f filter

Shift-invariant: shift(apply(I,f)) = apply(shift(I,f))

Intuitively: only depends on filter neighborhood



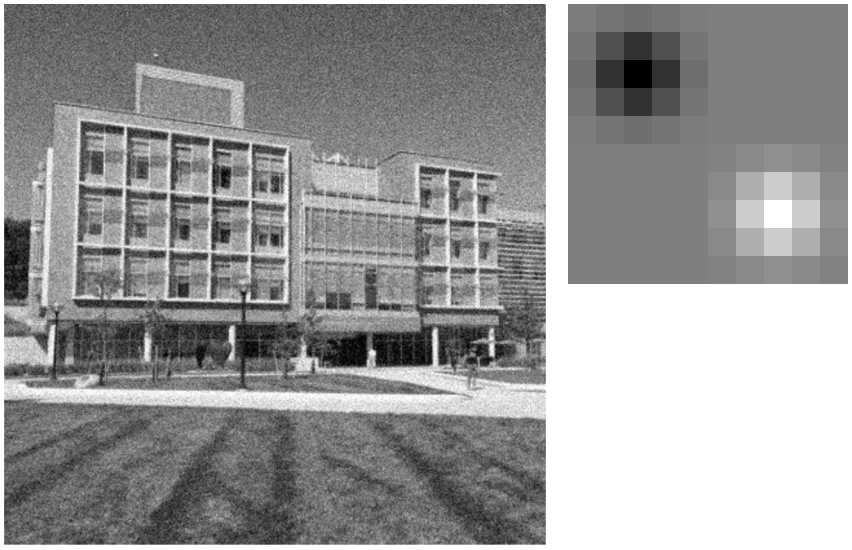


Painful Details – Signal Processing

What I just showed is often called “convolution”. *Actually* cross-correlation. Source of ***terrible*** confusion.

Cross-Correlation
(Original Orientation)

Convolution
(Flip filter in x,y first)



Convolution

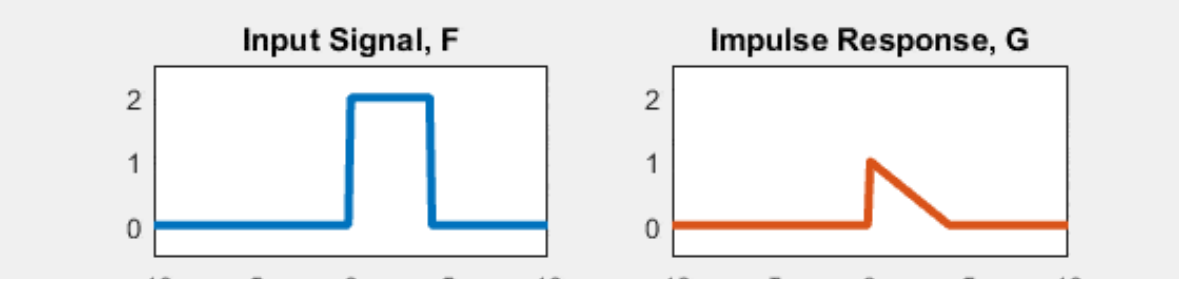
To be more clear:

```
def imageFilter2D(image, filter):
    for y in range(…): for x in range(…)
    # you'll fill this in
    return filtered

def imageConvolve2D(image, filter):
    # flip the filter left/right and up/down
    filterUse = np.fliplr(np.flipud(filter))
    return imageFilter2D(image, filterUse)
```

Convolutions

- $(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau]g[t-\tau]$



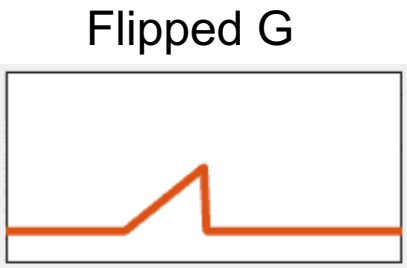


Image Credit: https://quincyaflint.weebly.com/academic-material/discrete-convolution

Convolutions

- $(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau]g[t-\tau]$

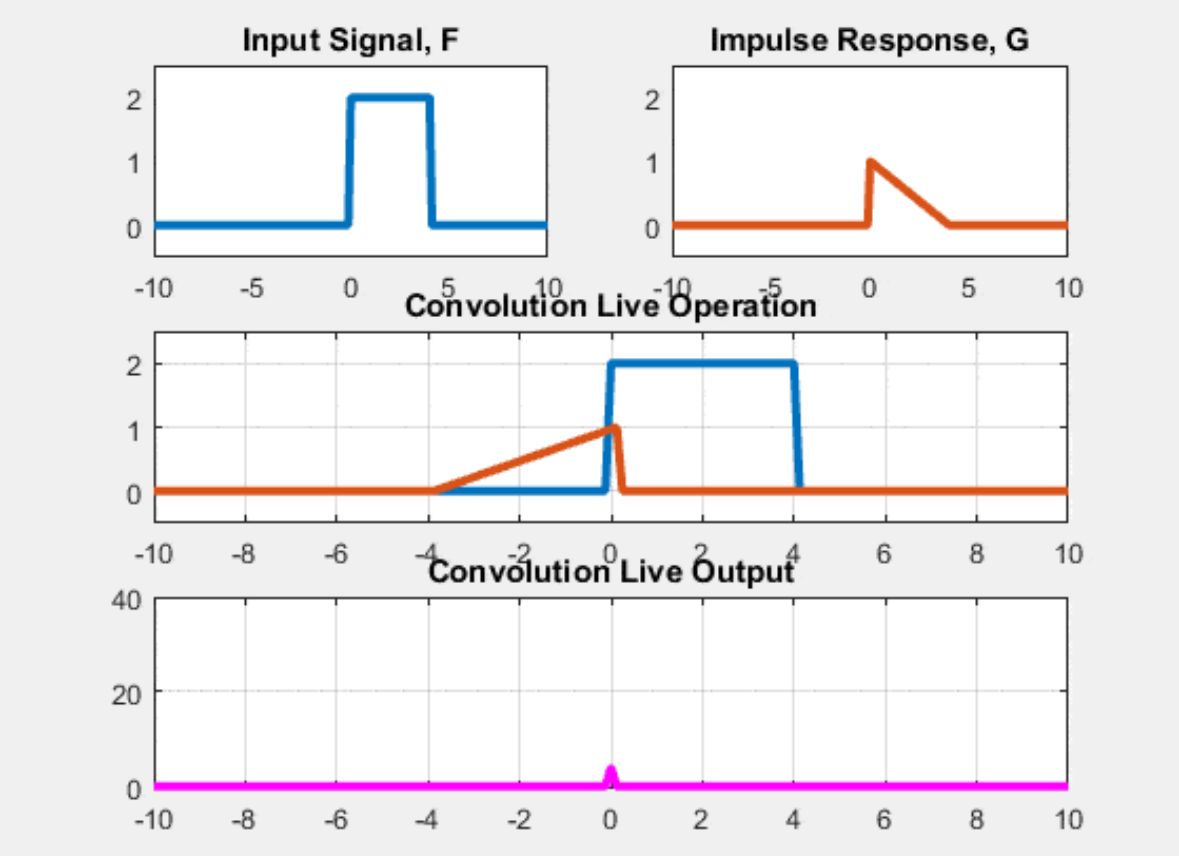


Image Credit: https://quincyaflint.weebly.com/academic-material/discrete-convolution

Convolutions is Commutative

- $(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau] g[t-\tau]$
- Replace τ with $t - \tau$:
- $(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[t-\tau] g[\tau] = (\mathrm{g} * \mathrm{f})[\mathrm{t}]$

Cross-Correlation is Not

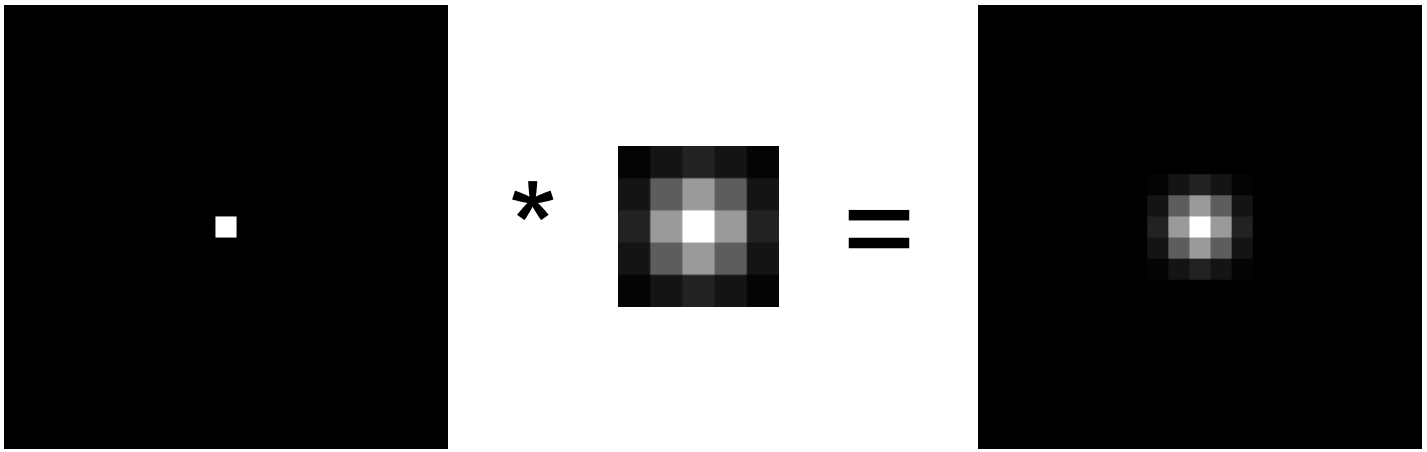
- $(f \star g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau] g[t+\tau]$
- Replace τ with $t + \tau$:
- $(f \star g)[t] = \sum_{\tau=-\infty}^{\infty} f[t+\tau] g[2t+\tau]$
  $\neq (\mathrm{g} \star \mathrm{f})[\mathrm{t}]$
- Not Commutative!

Convolution vs Cross-Correlation

- What we actually do is Cross-Correlation
- But they are very similar
- And Computer Vision people just use the term “Convolution.”

Properties of Convolution

- Convolution ($*$) is a shift-invariant linear operator
- Commutative: $f * g = g * f$
- Associative: $(f * g) * h = f * (g * h)$
- Distributes over +: $f * (g + h) = f * g + f * h$
- Scalars factor out: $kf * g = f * kg = k (f * g)$
- Identity (a single one with all zeros):



Property List: K. Grauman

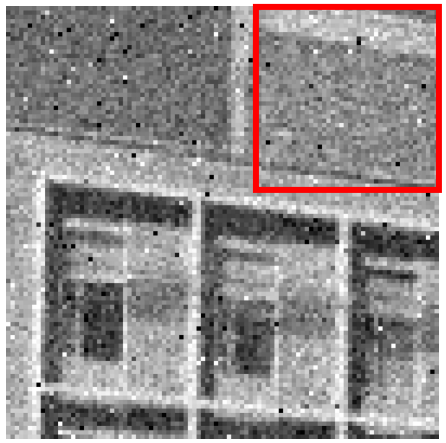
Questions?

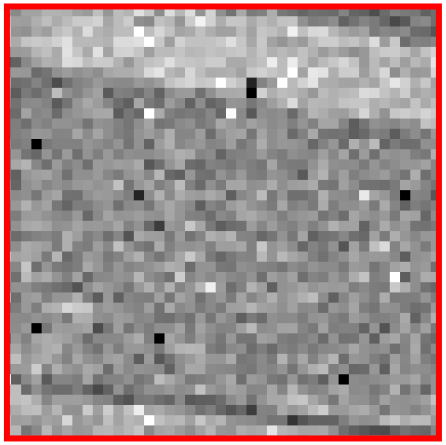
- Nearly everything onwards is a convolution.
- This is important to get right.

Smoothing With A Box

Intuition: if filter touches it, it gets a contribution.

Input

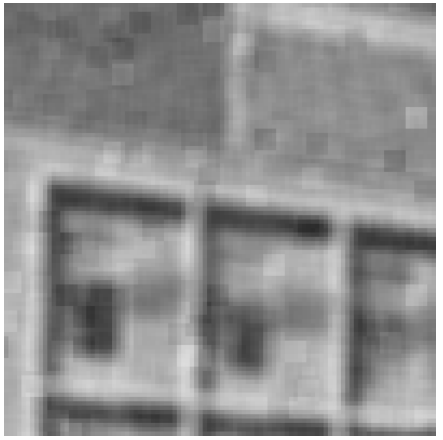


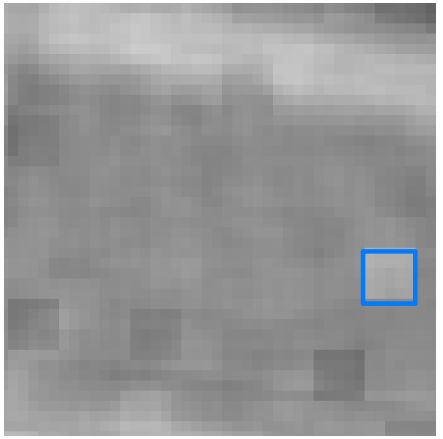


Filter

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Output





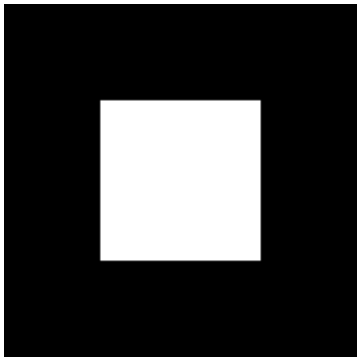
we don't like this boxy thing

We want to remove them

Solution – Weighted Combination

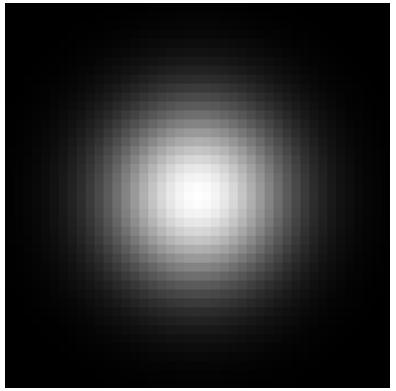
Intuition: weight according to closeness to center. Define 0,0 to be center pixel, then:

$$Filter_{ij} \propto 1$$



What's this?

$$Filter_{ij} \propto \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

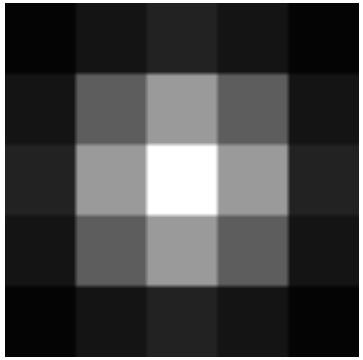


Recognize the Filter?

It's a Gaussian!

$$Filter_{ij} \propto \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

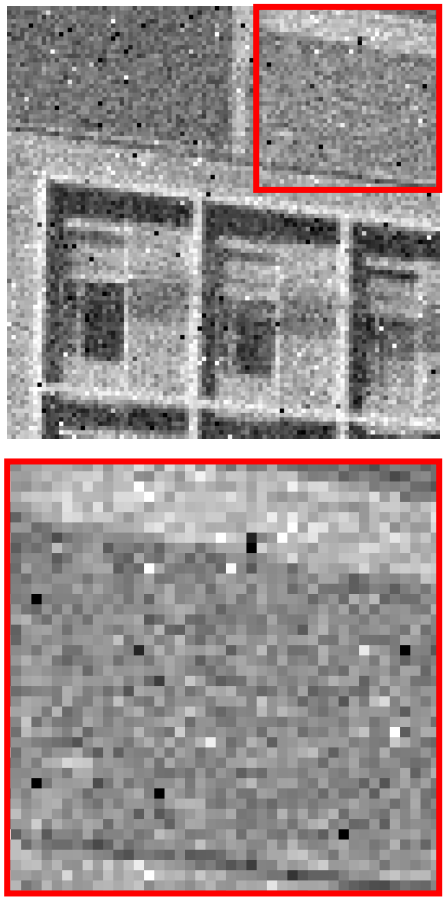
$$\begin{bmatrix} 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\ 0.013 & 0.060 & 0.098 & 0.060 & 0.013 \\ 0.022 & 0.098 & 0.162 & 0.098 & 0.022 \\ 0.013 & 0.060 & 0.098 & 0.060 & 0.013 \\ 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \end{bmatrix}$$



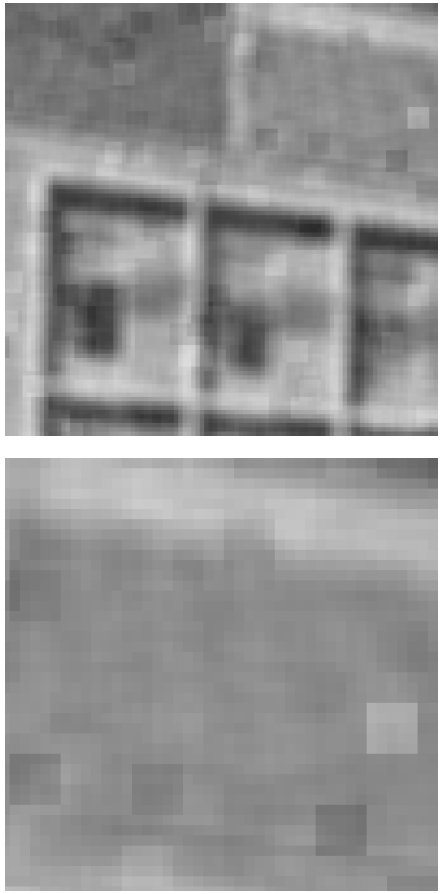
Smoothing With A Box & Gauss

Still have some speckles, but it's not a big box

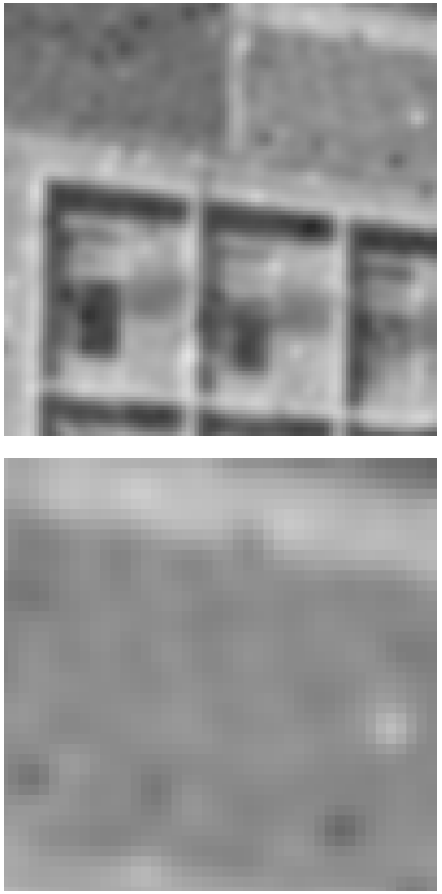
Input



Box Filter

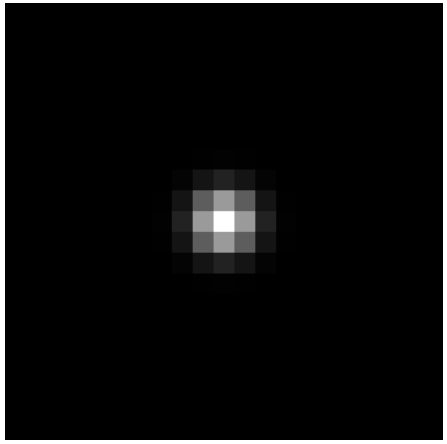


Gauss. Filter

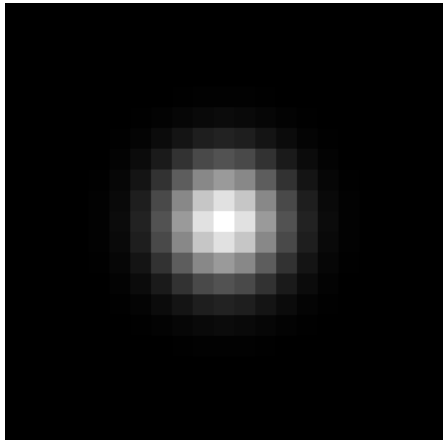


Gaussian Filters

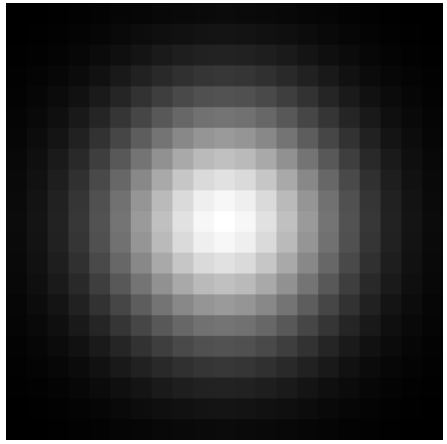
σ = 1
filter = 21x21



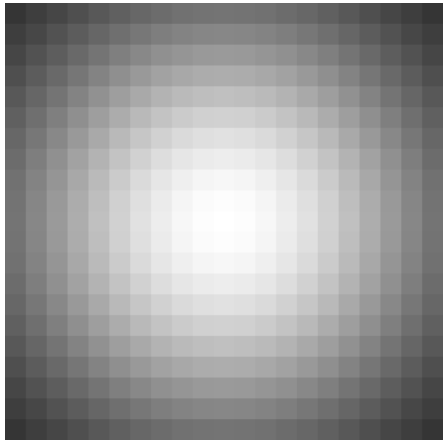
σ = 2
filter = 21x21



σ = 4
filter = 21x21

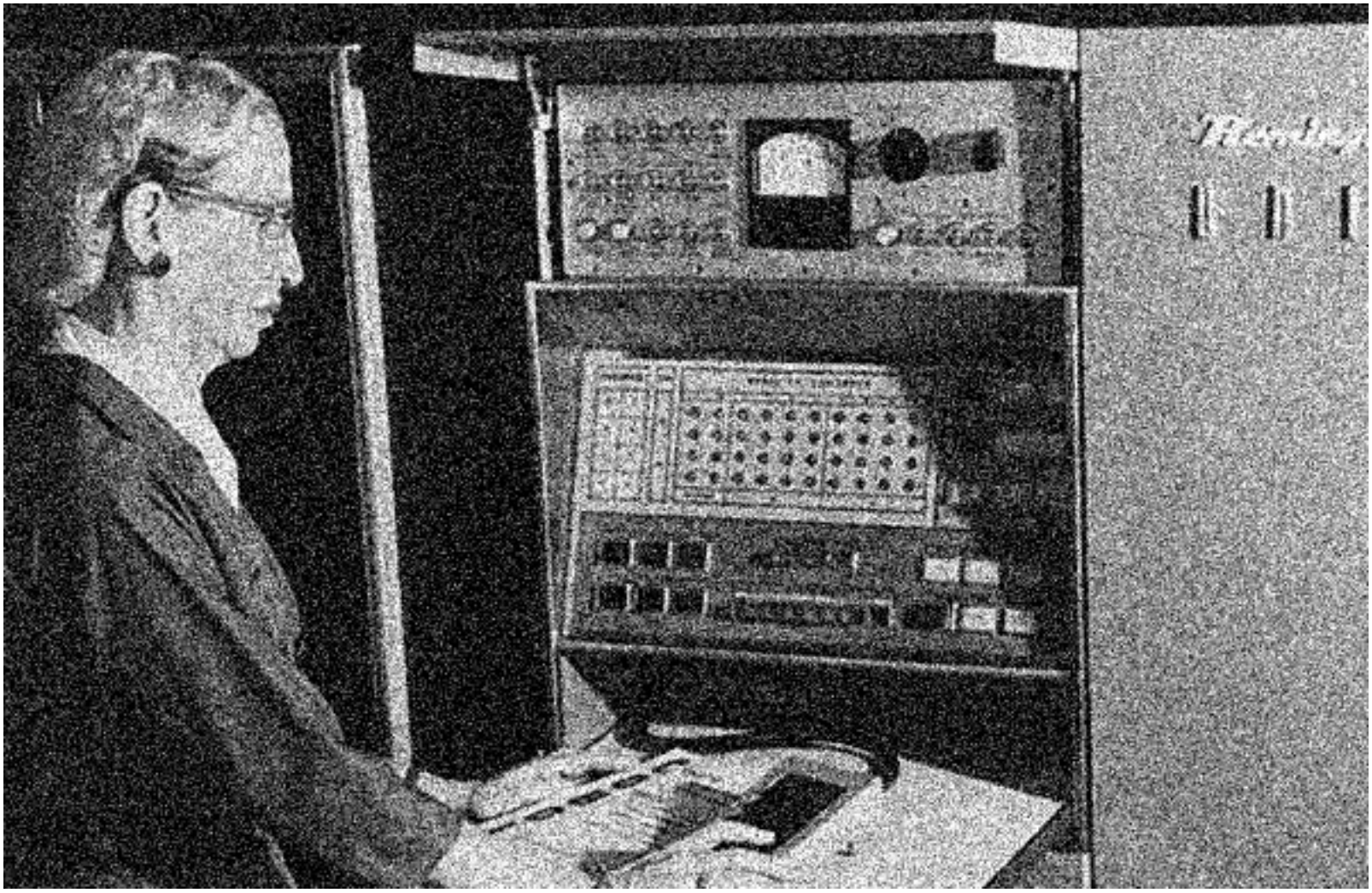


σ = 8
filter = 21x21



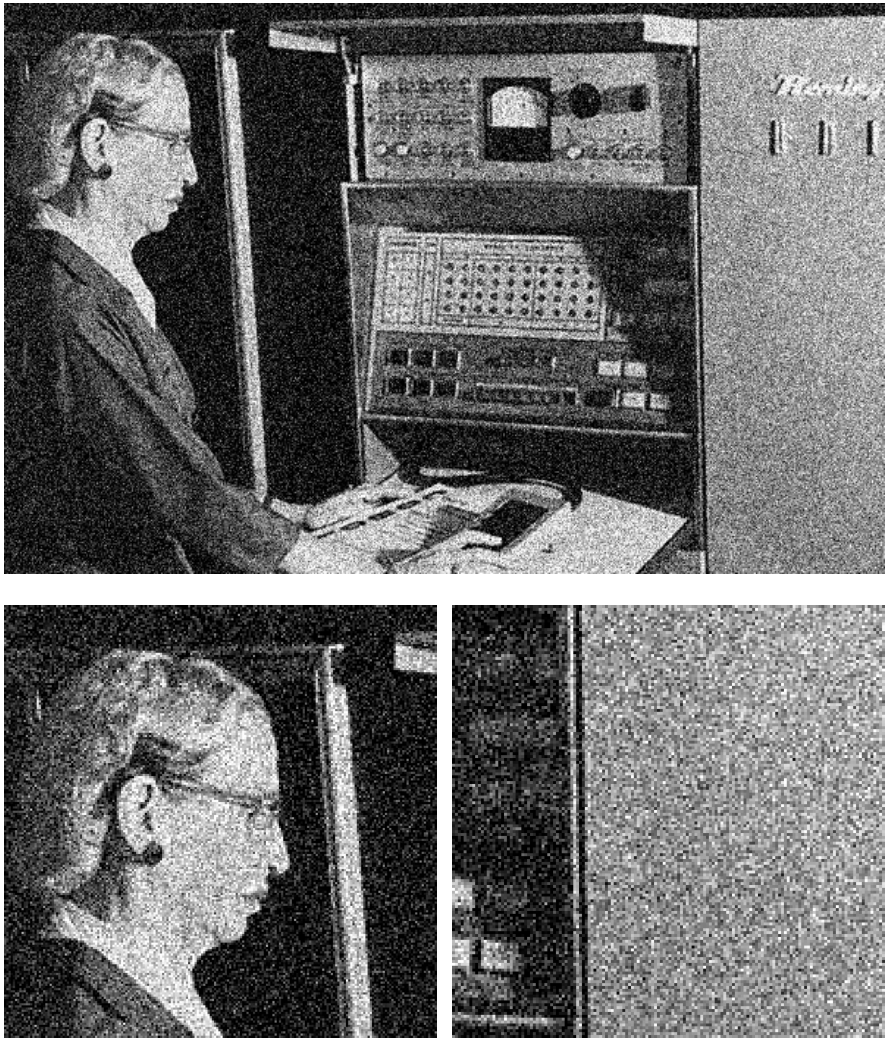
Note: filter visualizations are independently normalized throughout the slides so you can see them better

Applying Gaussian Filters



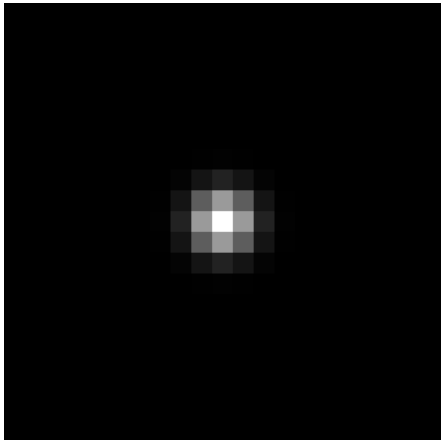
Applying Gaussian Filters

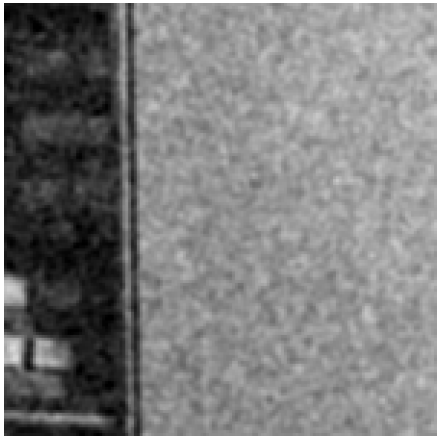
Input Image
(no filter)



Applying Gaussian Filters

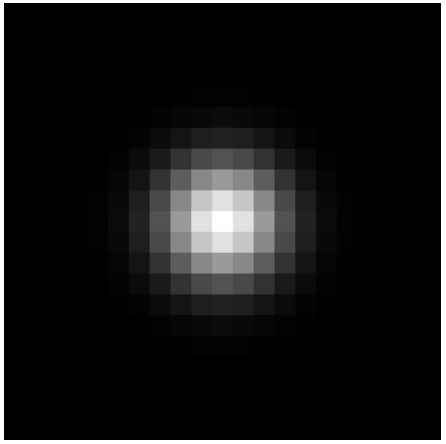
$\sigma = 1$

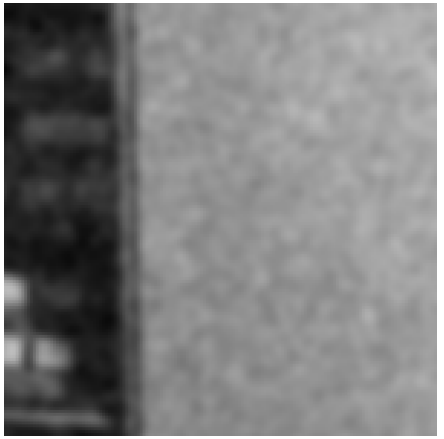




Applying Gaussian Filters

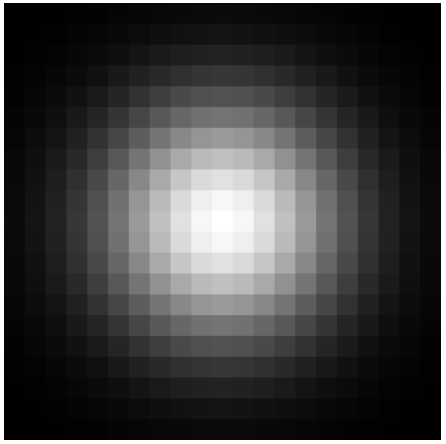
$\sigma = 2$

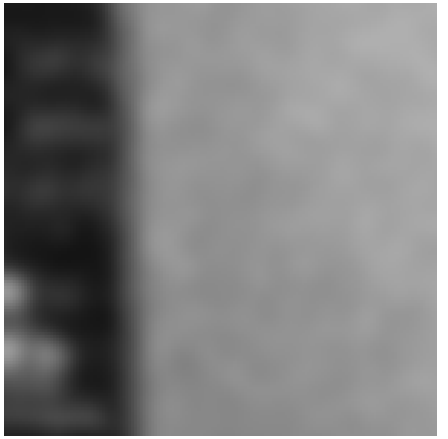




Applying Gaussian Filters

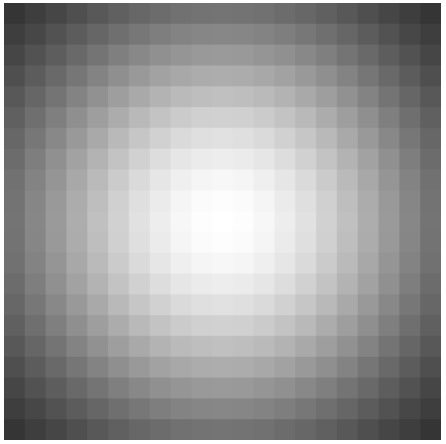
$\sigma = 4$





Applying Gaussian Filters

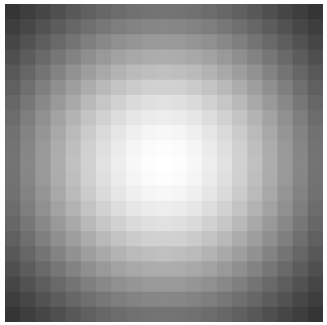
$\sigma = 8$



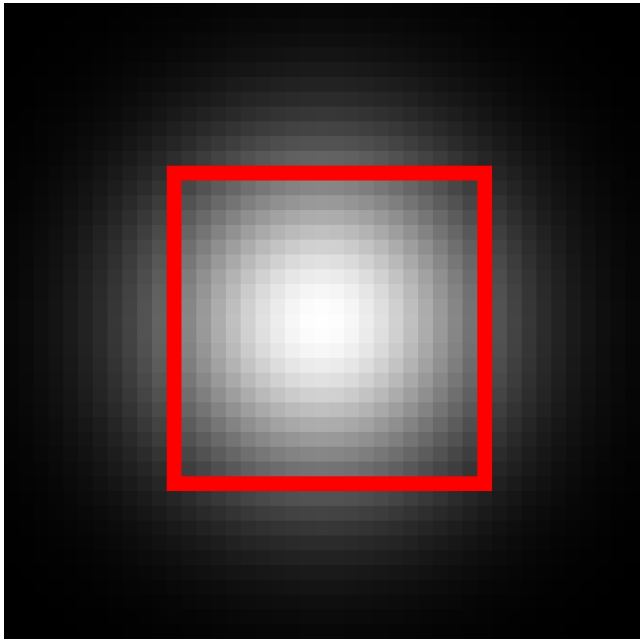
Picking a Filter Size

Too small filter → bad approximation
Want size ≈ 6σ (99.7% of energy)
Left far too small; right slightly too small!

σ = 8, size = 21



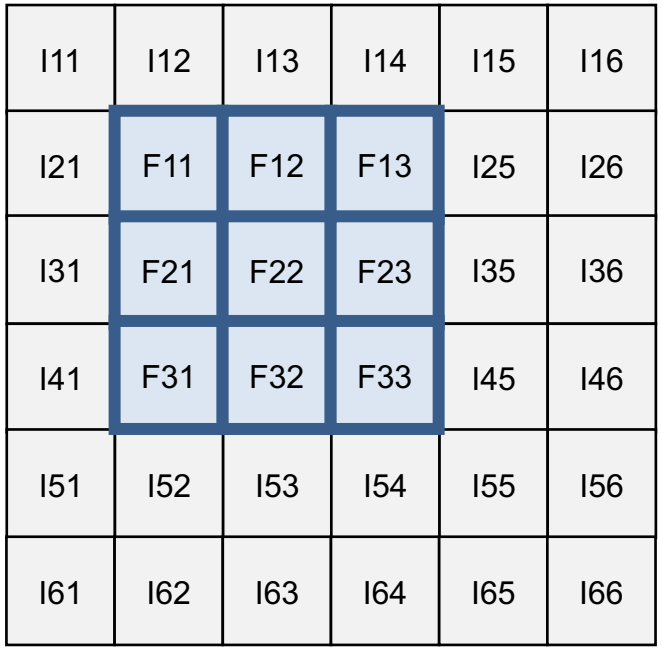
σ = 8, size = 43



Runtime Complexity

Image size = NxN = 6x6

Filter size = MxM = 3x3

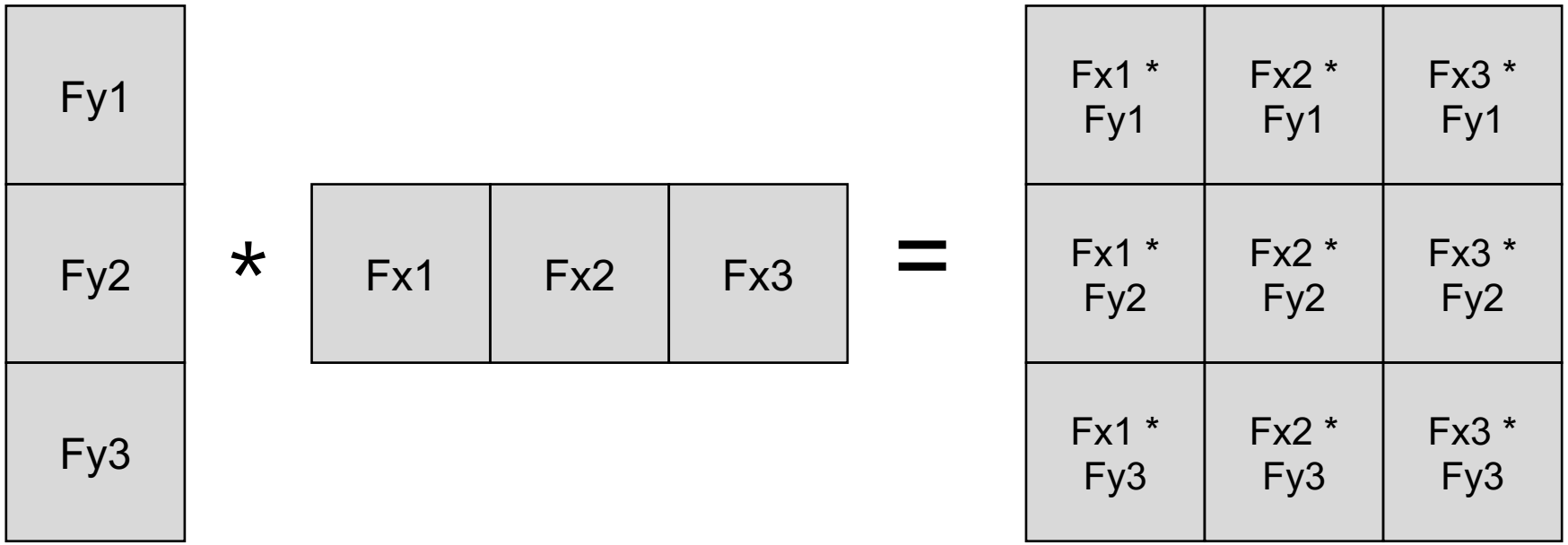



```
for ImageY in range(N):
    for ImageX in range(N):
        for FilterY in range(M):
            for FilterX in range(M):
                ...
```

Time: $O(N^2M^2)$

Separability

Conv(vector, transposed vector) → outer product



Separability

$$Filter_{ij} \propto \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

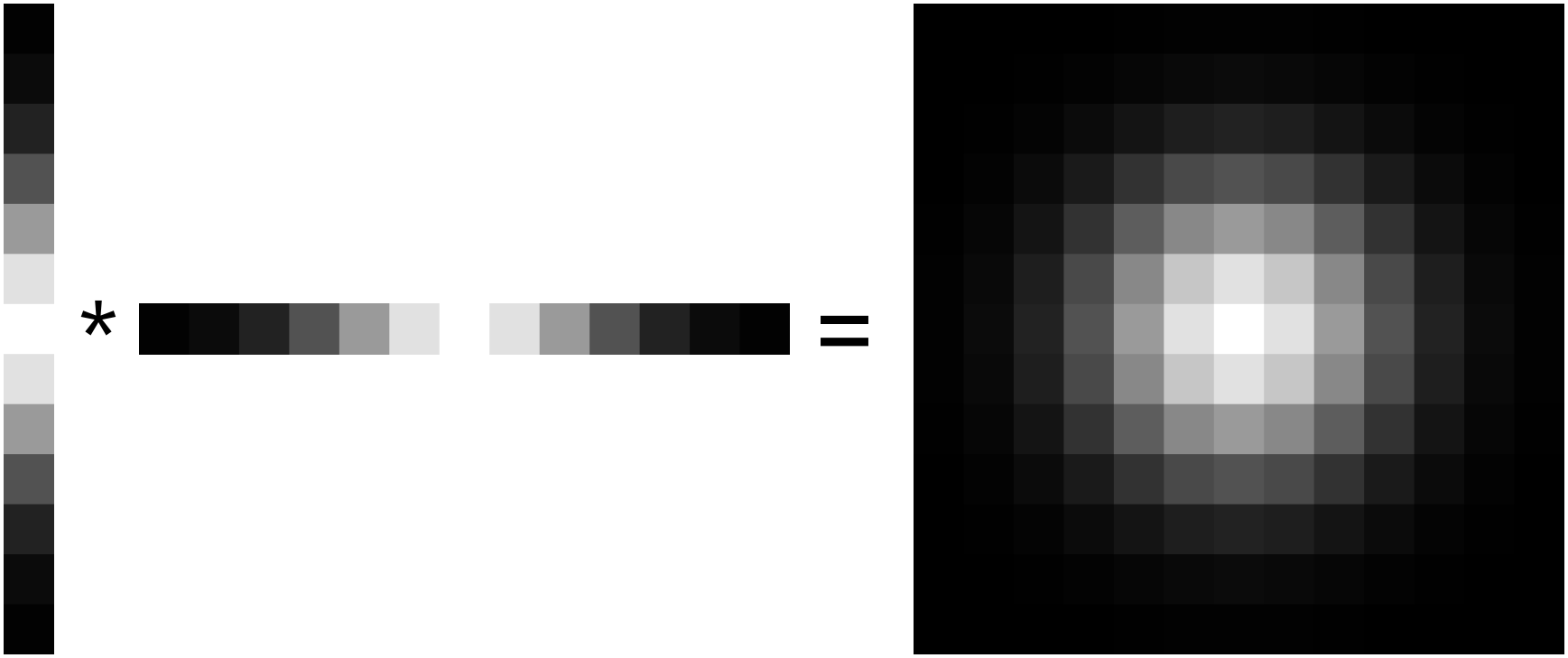
$$\longrightarrow$$

$$Filter_{ij} \propto \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right)$$

Separability

1D Gaussian $*$ 1D Gaussian = 2D Gaussian

Image $*$ 2D Gauss = Image $*$ (1D Gauss $*$ 1D Gauss)
= (Image $*$ 1D Gauss) $*$ 1D Gauss



Runtime Complexity

Image size = NxN = 6x6

Filter size = Mx1 = 3x1

I11	I12	I13	I14	I15	I16
I21	F1	I23	I24	I25	I26
I31	F2	I33	I34	I35	I36
I41	F3	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56
I61	I62	I63	I64	I65	I66

What are my compute savings for a 13x13 filter?

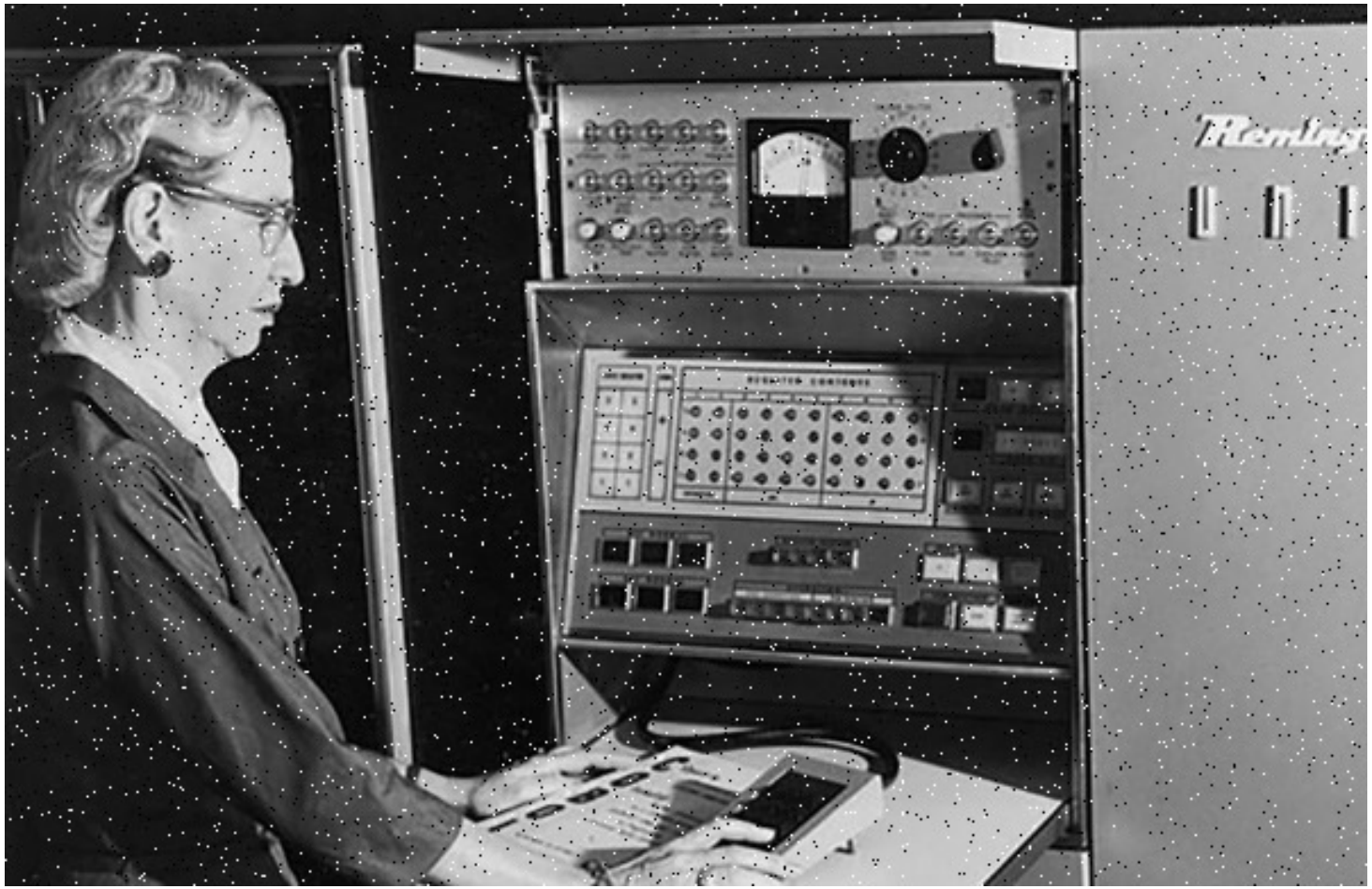
```
for ImageY in range(N):
    for ImageX in range(N):
        for FilterY in range(M):
            …
for ImageY in range(N):
    for ImageX in range(N):
        for FilterX in range(M):
            …
```

Time: $O(N^2M)$

Why Gaussian?

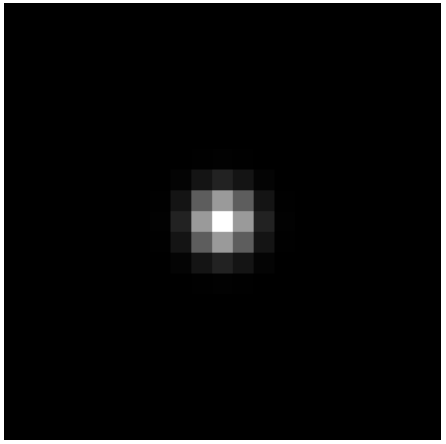
Gaussian filtering removes parts of the signal above a certain frequency. Often noise is high frequency and signal is low frequency.

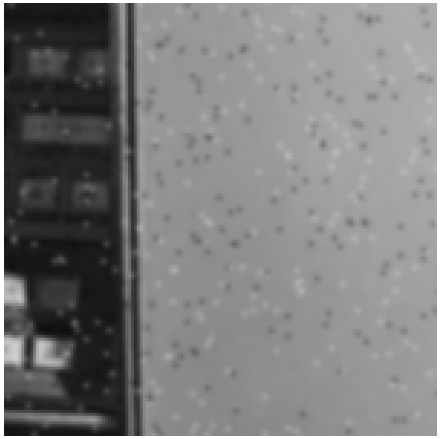
Where Gaussian Fails



Applying Gaussian Filters

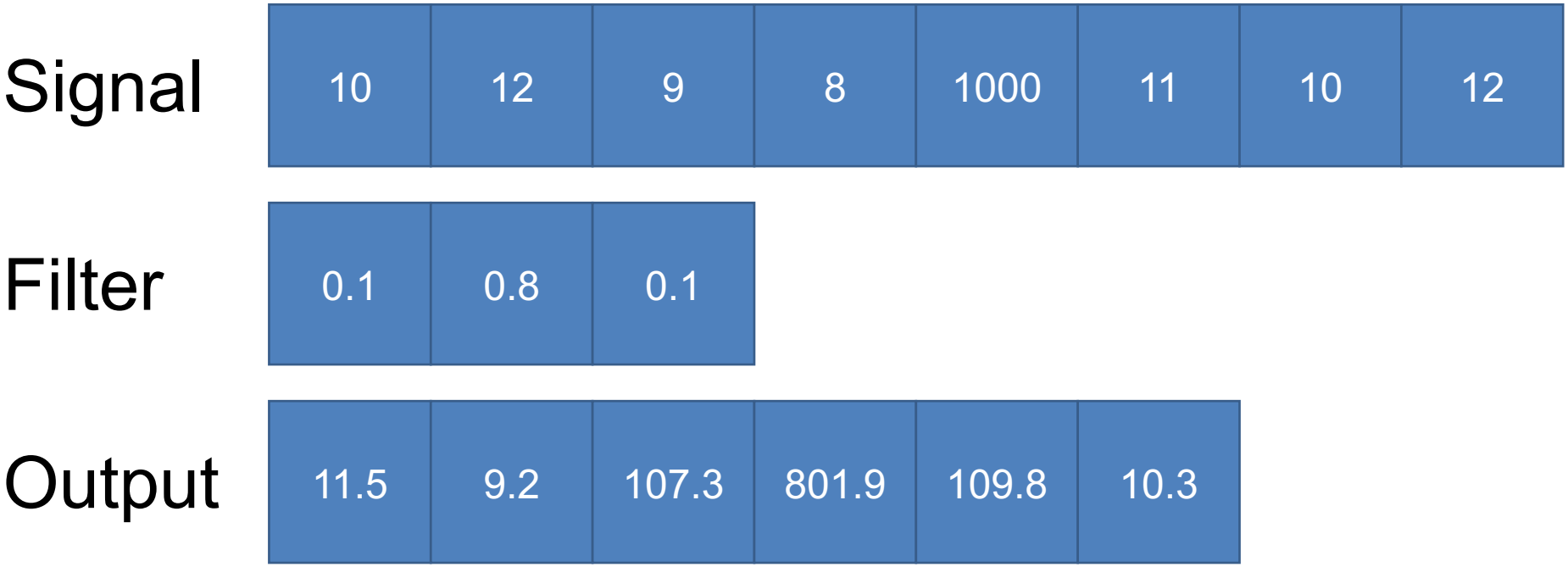
$\sigma = 1$





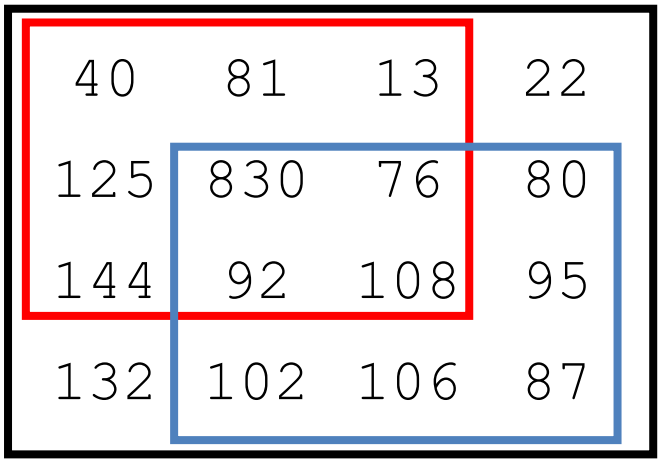
Why Does This Fail?

Means can be arbitrarily distorted by outliers



What else is an “average” other than a mean?

Non-linear Filters (2D)



[040, 081, 013, 125, 830, 076, 144, 092, 108]

Sort

[013, 040, 076, 081, 092, 108, 125, 144, 830]

92

[830, 076, 080, 092, 108, 095, 102, 106, 087]

Sort

[076, 080, 087, 092, 095, 102, 106, 108, 830]

95

Applying Median Filter

Median Filter (size=3)

Applying Median Filter

Median Filter (size = 7)

Is Median Filtering Linear?

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

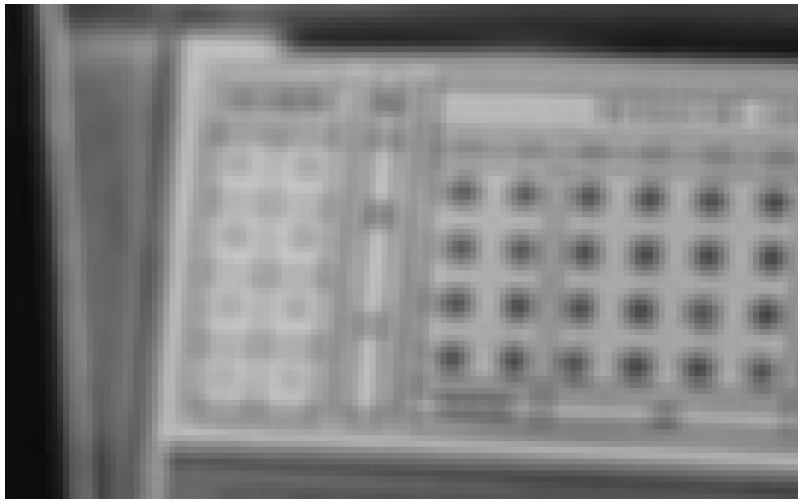
$$1 + 0 = 2$$

Example from (I believe): Kristen Grauman

Bilateral Filtering

- Gaussian Filtering
- Pro: Removes noise
- Con: Removes details (blurry output)

- Alternative:
- Bilateral Filtering!

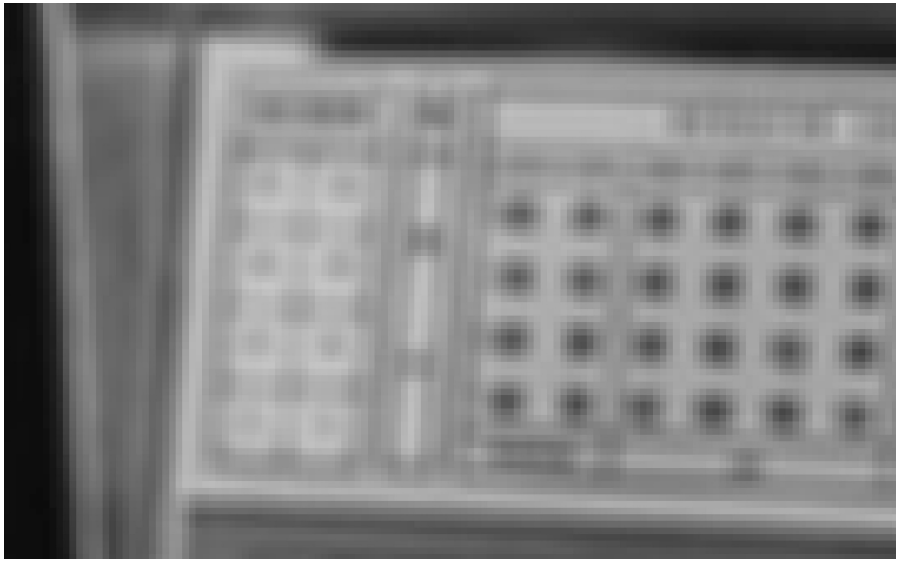


Bilateral Filtering

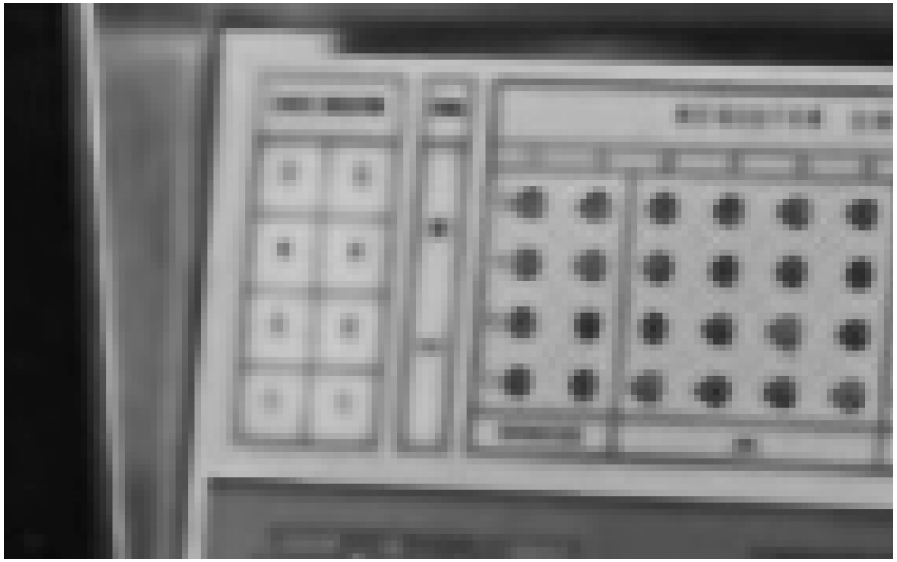
- Gaussian Filtering
- $I^G(x) = \frac{1}{W}\sum_{x_i \in \Omega} I(x_i) g_\sigma(||x_i - x||)$
- Bilateral Filtering:
  - Weight the kernel based on content similarity!
  - Preserves Edges!
- $I^B(x) = \frac{1}{W}\sum_{x_i \in \Omega} I(x_i) g_\sigma\left(\left|\left|x_i - x\right|\right|\right) f_\eta(||I(x_i) - I(x)||)$

Bilateral Filtering

- Bilateral Filtering Preserves Edges!
- Better edges and texture than Gaussian or Median
- Computationally Heavy



Gaussian Filtering

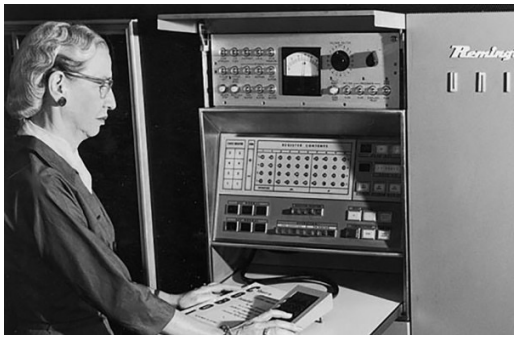


Bilateral Filtering

Some Examples of Filtering Effects

Filtering – Sharpening

Image

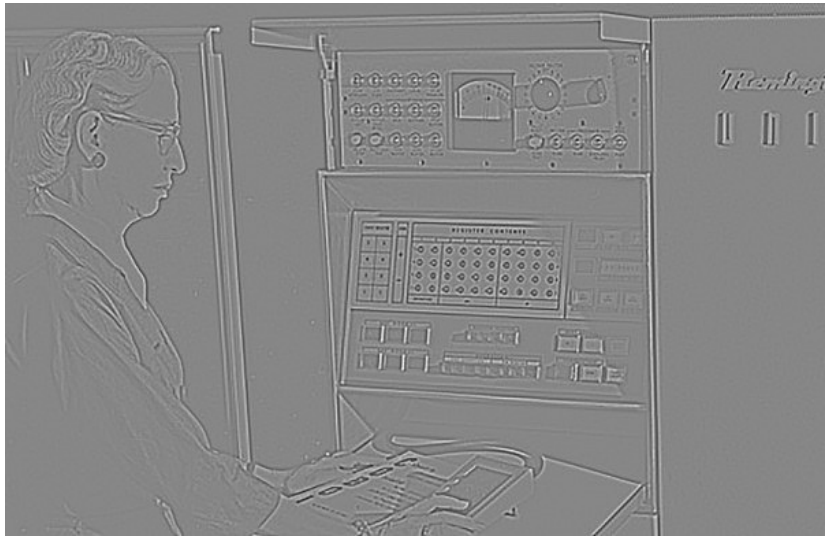


-

Smoothed

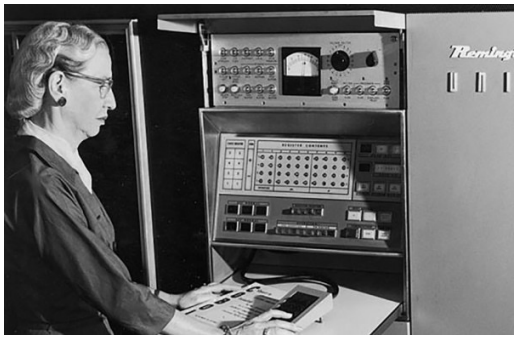
Details

=



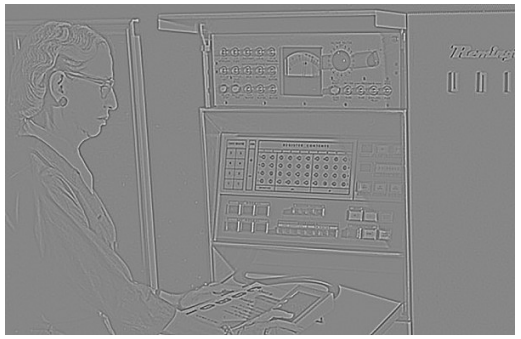
Filtering – Sharpening

Image



+α

Details

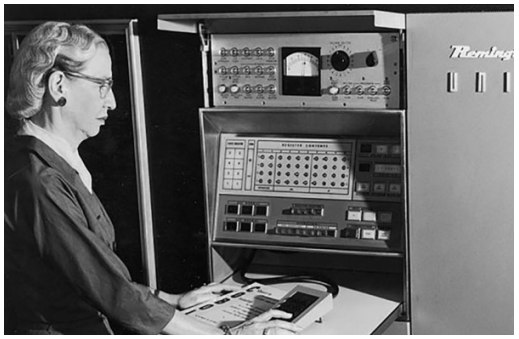


“Sharpened” α=1

=

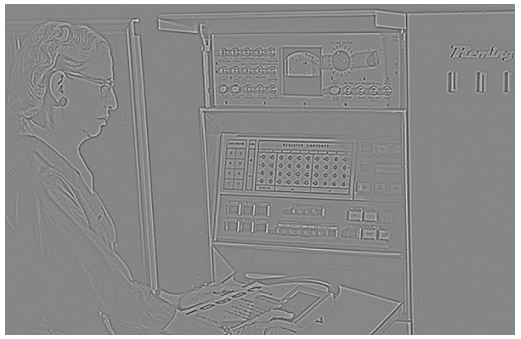
Filtering – Sharpening

Image



+α

Details

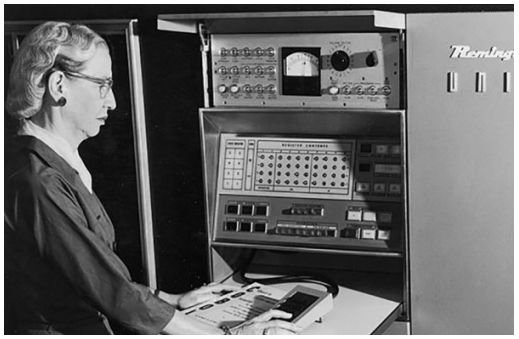


“Sharpened” α=0

=

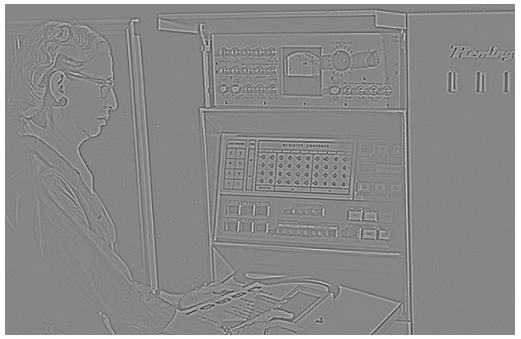
Filtering – Sharpening

Image



+α

Details

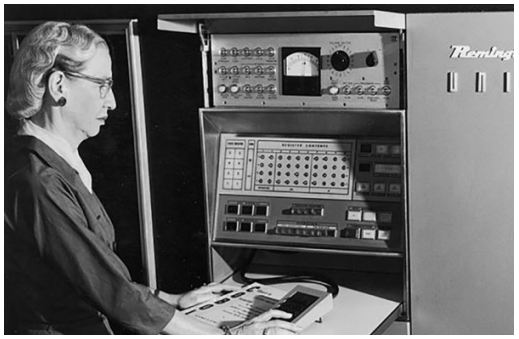


“Sharpened” α=2

=

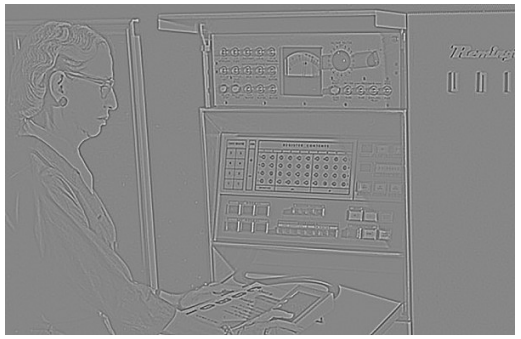
Filtering – Sharpening

Image



+α

Details

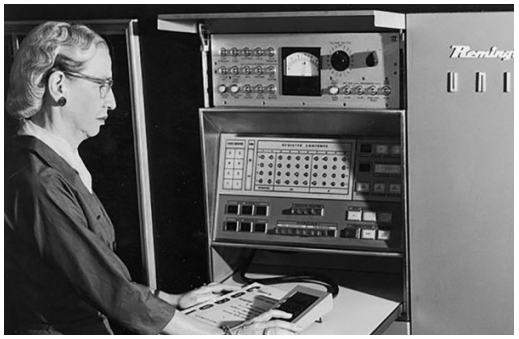


“Sharpened” α=0

=

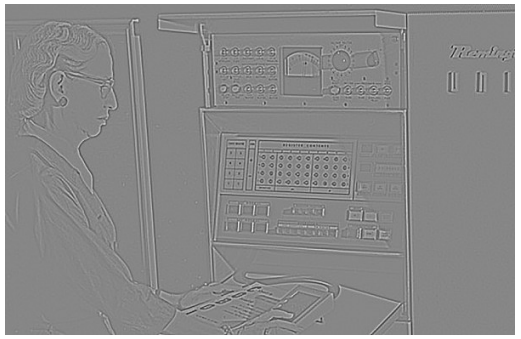
Filtering – Extreme Sharpening

Image



+α

Details



“Sharpened” α=10

=

Filtering

What's this Filter?

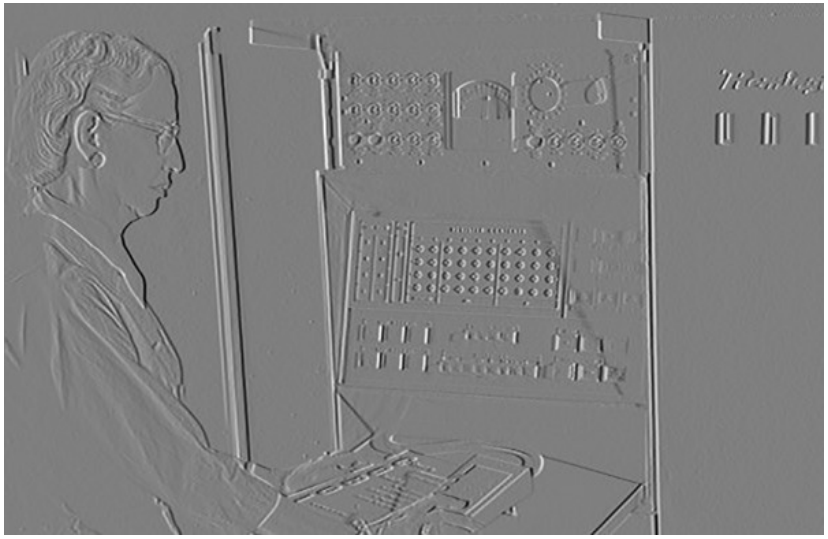
-1	0	1

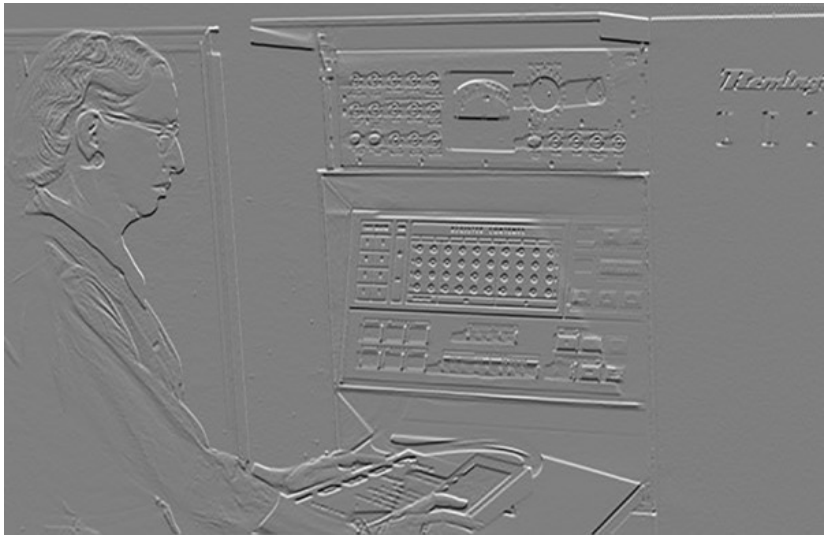
Dx

-1	0	1

^{T}

Dy





Filtering – Derivatives

$$(Dx^2 + Dy^2)^{1/2}$$

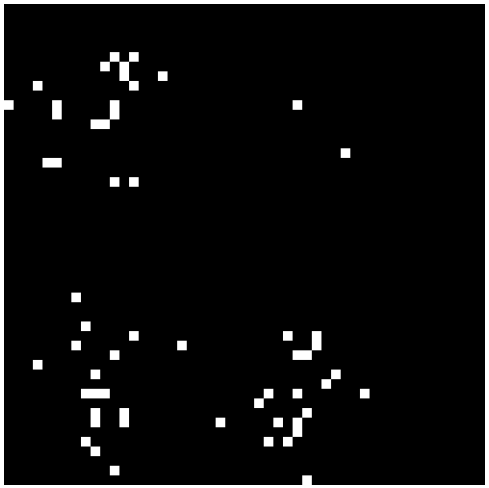
Filtering – Bonus

- If you're curious, you can use filters to accomplish a surprisingly large number of things.

Filtering – Counting

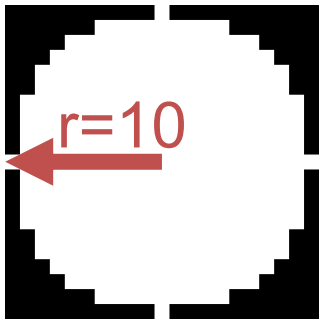
How many “on” pixels have
10+ neighbors within 10 pixels?

Pixels



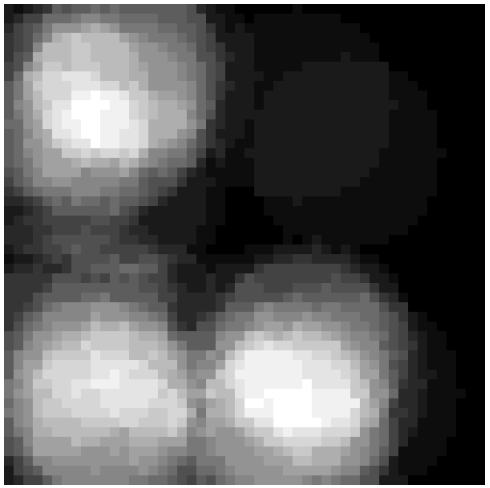
*

Disk



=

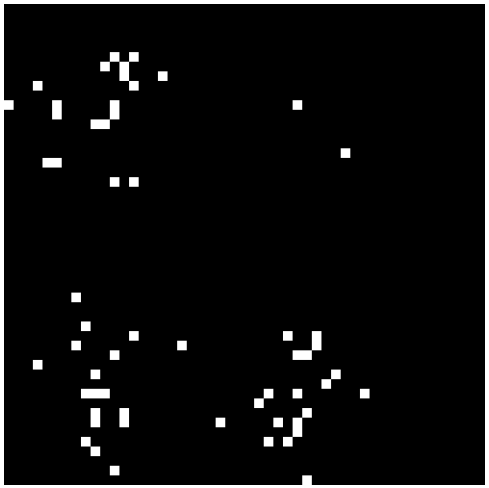
???



Filtering – Counting

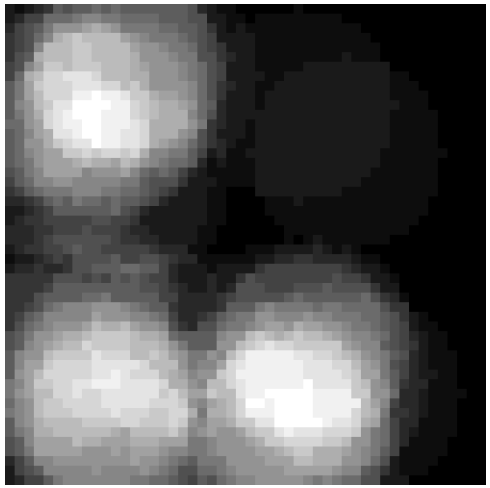
How many “on” pixels have
10+ neighbors within 10 pixels?

Pixels



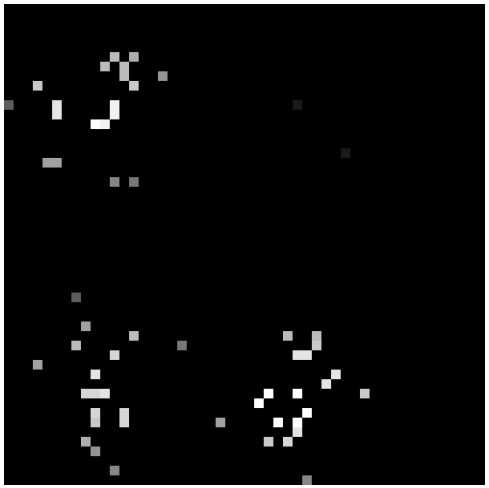
x

Density



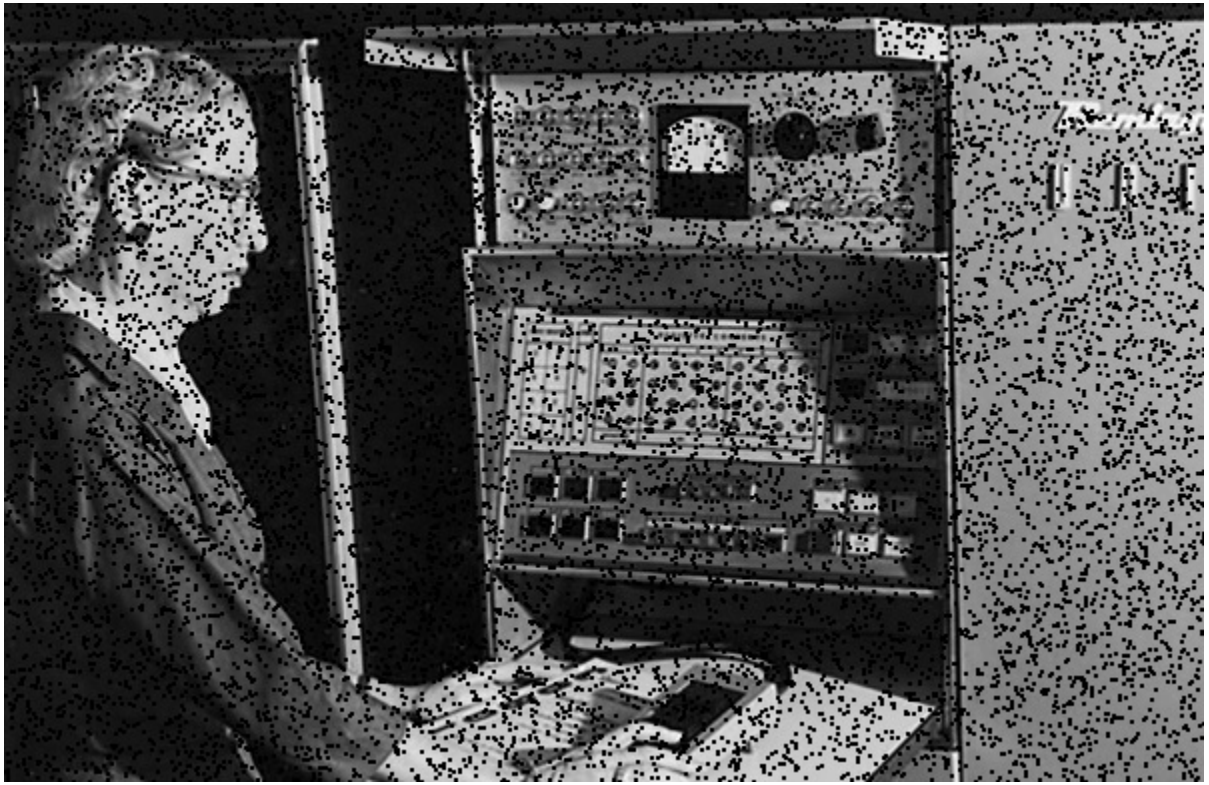
=

Answer



Filtering – Missing Data

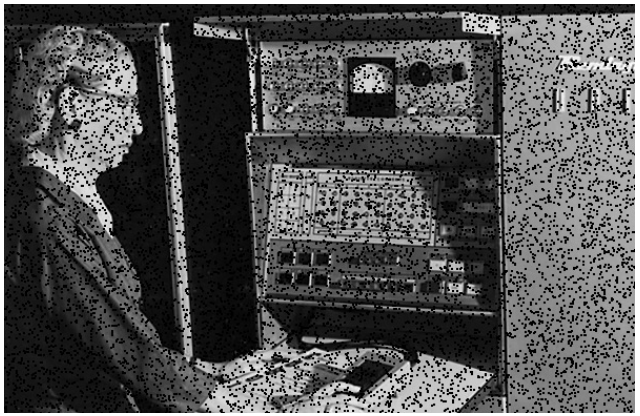
Oh no! Missing data!
(and we know where)



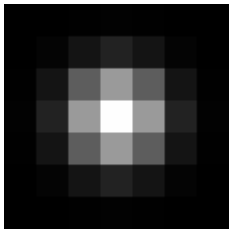
Common with many non-normal cameras (e.g., depth cameras)

Filtering – Missing Data

Image

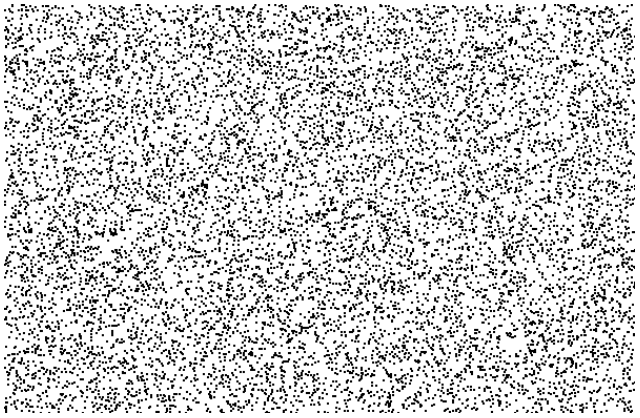


*

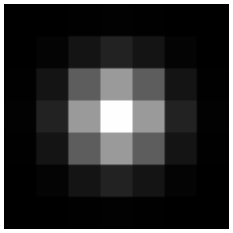


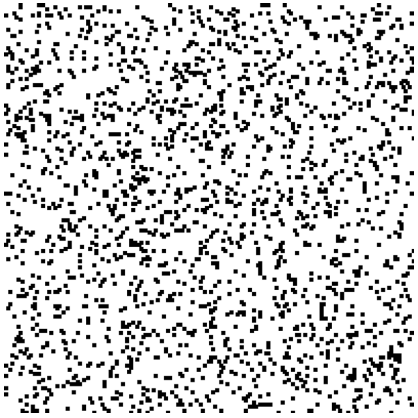
Per-element /

Binary Mask



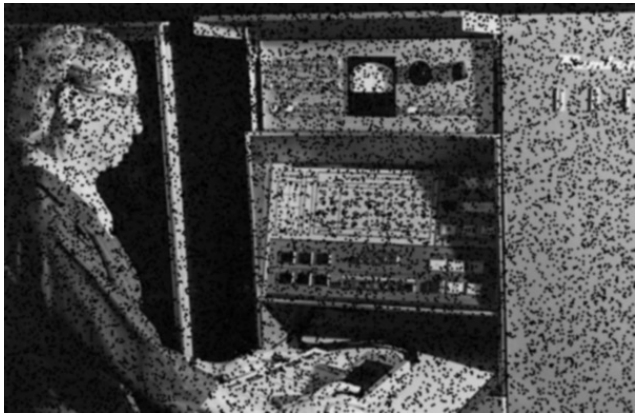
*



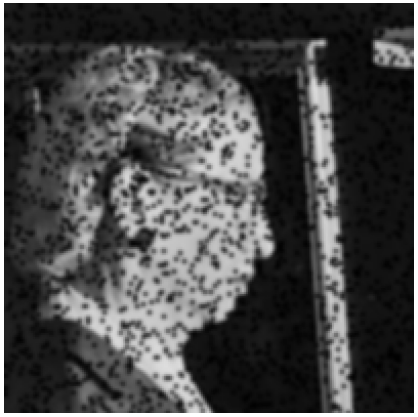


Filtering – Missing Data

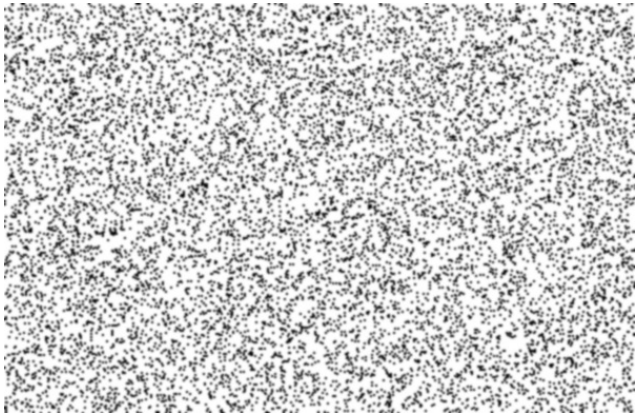
Image

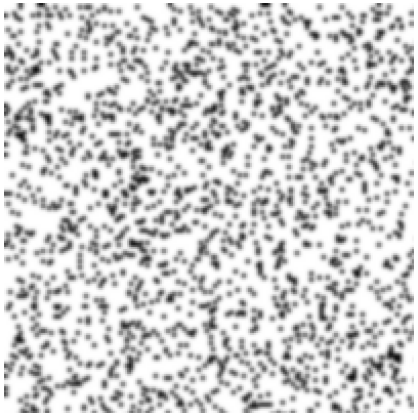


Per-element /



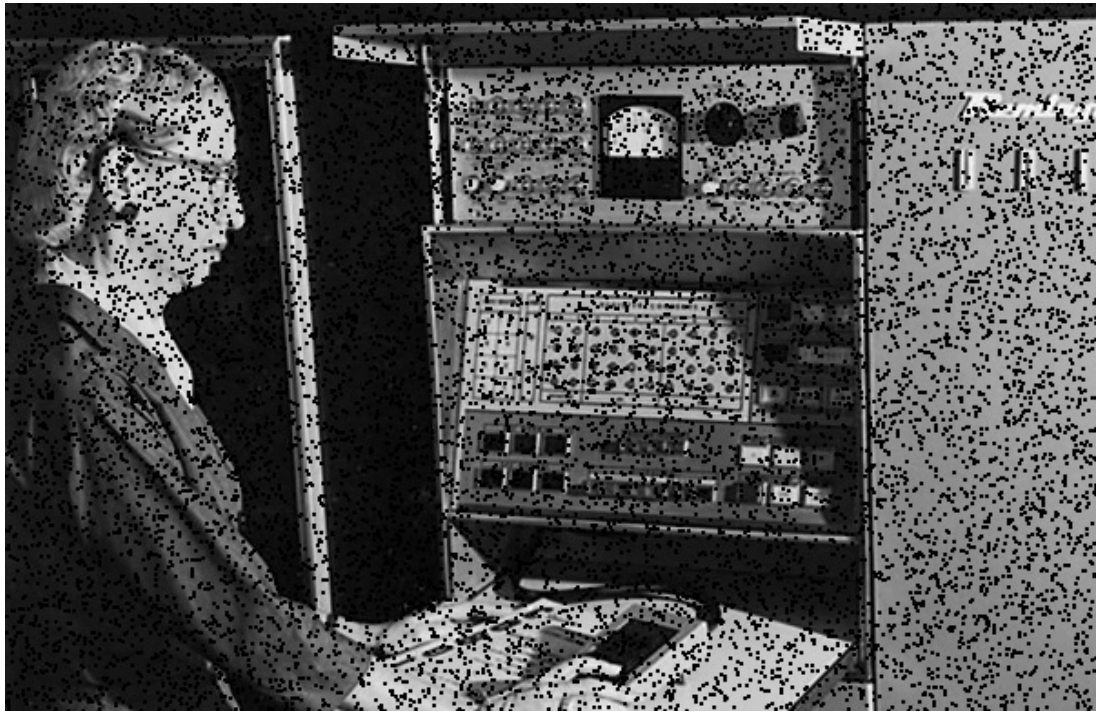
Binary Mask

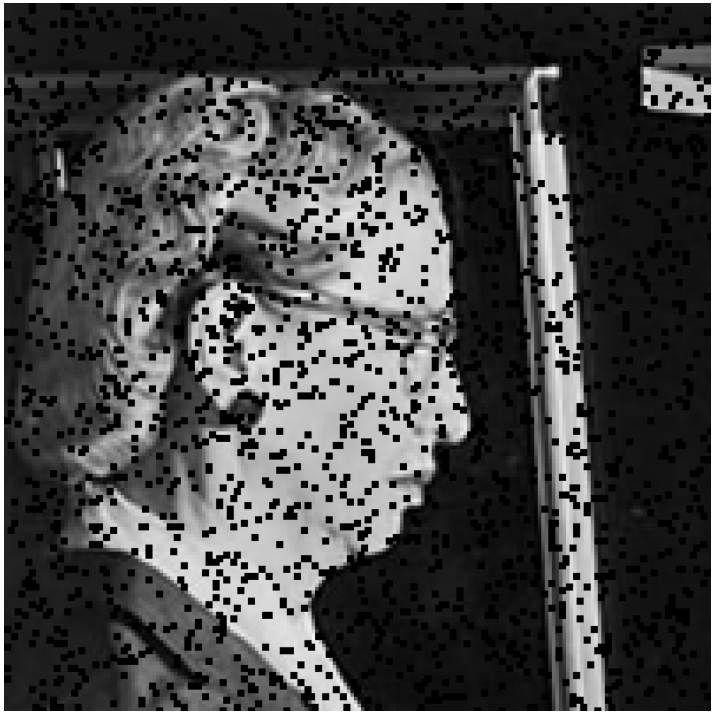




Filtering – Missing Data

Before





Filtering – Missing Data

After

Filtering – Missing Data

After (without missing data)